VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozpoznávání textu a technologie OCR
Text Recognition by Using OCR

Ostrava 2010 Bc. Milan Chaloupka

Zadání …

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Kopřivnici 3.května 2010

……
Milan Chaloupka

Poděkování

Děkuji tímto vedoucímu diplomové práce Ing. Radoslavu Fasugovi, Ph.D. za odbornou pomoc a konzultaci při realizaci tohoto projektu.

Abstrakt a klíčová slova

Abstrakt:

Tato práce řeší problematiku vad a nedokonalostí vstupních materiálů pro OCR a obsahuje popis možného řešení pro problematiku rozpoznání tištěného textu. Obsahem práce je popis základních algoritmů pro úpravu vstupních materiálů, jakožto algoritmy pro detekci hran v obraze, segmentace barev v obraze a vyhlazování nekvalitně získaného obrazu. V práci je také provedeno zhodnocení použitých algoritmů a porovnání získaných výsledků. V poslední řadě je vysvětlena problematika rozpoznávání prohnutého textu a textu opsaného na kružnici s možností řešení tohoto problému.

Klíčová slova:

OCR, RGB, šedotónový obraz, prahování, globální prahování, Otsu prahování, p-procentní prahování, detekce hran, operátor Prewittové, Sobelův operátor, Laplacian of Gaussian, invarianty, Fourierovy deskriptory, momenty, normální momenty, centrální momenty, sedm Huových invariantních momentů, Zernikovy momenty, Pseudo-Zernikovy momenty, text na kružnici, prohnutý text, JPG, PDF, C#.

Abstract and keywords

Abstract:

This thesis solves the problem of defects and imperfections in the input materials for the OCR and describes possible solutions for issues of recognition of printed text. The thesis is a description of the basic algorithms for the adjustment of raw materials, as algorithms for edge detection in image, color segmentation in image and smoothing produced poor image. The thesis is also an assessment of the algorithms and compare the results obtained. Finally, the issue is explained recognition of curved text and text in circle with the possibility of solving this problem.

Keywords:

OCR, RGB, grayscale image, thresholding, global thresholding, Otsu thresholding, p-percentage thresholding, edge detection, Prewitt operator, Sobel operator, Laplacian of Gaussian, invariants, Fourier descriptors, moments, normal moments, central moments, Hu´s seven moment invariants, Zernike moments, Pseudo-Zernike moments, text in circle, curved text, JPG, PDF, C#.

Seznam použitých symbolů a značek

PC	Personal Computer – osobní počítač
OCR	Optical Character Recognition – optické rozpoznávání znaků
ICR	Intelligent Character Recognition – inteligentní rozpoznávání znaků
USPS	The United States Postal Service
GPO	General Post Office
PDA	Personal Digital Assistant
C#	Objektově orientovaný programovací jazyk vyvinutý firmou Microsoft
RGB	Barevný model RGB
DFT	Diskrétní Fourierova Transformace
XHTML	eXtended Hyper Text Meta Language

Obsah

1. Úvod

S rychlým rozvojem technologií a se zvyšujícím se požadavkem k digitalizaci tištěného textu vznikají nejrůznější nástroje k realizaci tohoto problému. V současnosti dochází k postupnému vytlačování tištěného textu a jeho náhrada za text digitální, který má značné výhody oproti textu na papíře. Mezi přednosti digitálního textu patří jeho přenositelnost, jednoduchá úprava, možnost text několikrát vytisknout apod. V této době si už nejde běžný den ani představit bez použití digitálního textu. Ať už v podobě e-mailu, SMS zpráv, nebo při čtení článků na internetu.

Nyní se objevuje další problém a to převod starších tištěných textů do elektronické podoby. Dříve se toto řešilo ručně, takže uživatel musel celý text sám kompletně přepsat, ale s narůstajícím požadavkem po této digitalizaci se takový postup jeví jako nereálný. Existují nástroje, které se touto problematikou zabývají, ale často řeší pouze určitou problematiku převodu tištěného textu do elektronické podoby a ne všechny požadavky, které by vyhovovaly všem skupinám uživatelů.

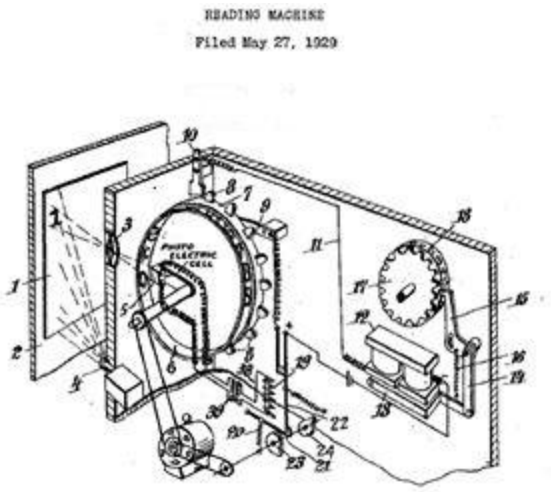
Převod textu do elektronické podoby zahrnuje několik důležitých fází. Tou vůbec první a velmi důležitou fází pro další pokračování je nutnost získat zdrojový text, dále obraz, který se bude digitalizovat. Tento problém se řeší různými způsoby. Ať už oskenováním nebo získáním z jiného zdroje. Samotný obraz pak nebývá příliš vhodný pro další převod. Může být různě barevný, obsahovat text v barevném poli, zakřivený text, šum a různé faktory, které zhoršují převod. Z tohoto důvodu je nutné obraz nejprve předzpracovat pro samotnou fázi optického zpracování znaků (OCR). Nakonec je předzpracovaný obraz použit jako vstupní obraz pro OCR. Výstupem fáze OCR je již digitalizovaný text, který se později může dále upravovat.

1.1 Historie

Historie OCR sahá až do roku 1929, kdy si rakouský vynálezce Gustav Tauschek nechal v Německu patentovat tzv. „Čtecí stroj“ (Reading Machine). Z tohoto patentu vycházejí základní koncepty dnešních OCR. Princip stroje byl založen na tzv. shodě šablon. Přístroj totiž obsahoval šablony jednotlivých znaků. Pokud se tedy šablona pěkně překrývala s daným znakem, prohlásil systém tyto znaky za shodné. Podobný princip se ještě dnes používá v některých aplikacích. Ještě tentýž rok koupila od pana Tauscheka tento patent americká firma IBM.

Pět let po konci druhé světové války pracoval americký kryptoanalytik David H. Shepard, který mimo jiné prolomil kód z japonského šifrovacího stroje Purple, na automatizaci přepisování dat do strojové formy. Bylo ale nutné převést do strojů velké množství tištěného textu. Proto se Shepard rozhodl, že s kolegou vyvine metodu pro strojové převádění tištěného textu do upravovatelné elektronické podoby. A tak během jednoho roku s přítelem Harveyem Cookem stavěli v podkroví systém, který si o rok později nechali patentovat pod jednoduchým názvem „Aparát na čtení“ (Apparatus for reading).

O pár let později si Shepard společně s Williamem Lawlessem založil firmu Intelligent Machines Research Corporation. Společně totiž chtěli komerčně uspět s jejich strojem „Gismo“. To se jim povedlo, protože se na scéně opět objevila firma IBM, která jejich patenty odkoupila. Dokonce jim taky zadala jednu zajímavou zakázku. Přesný úkol od IBM zněl: „Zkuste vyrobit stroj, který by byl schopný rozpoznávat ručně psané číslice.“ Mezitím IBM nadále rozvíjela systém rozpoznávání, a poprvé jej také pojmenovala slovy *Optical Character Recognition*. Tento název se udržel dodnes, přestože v dnešní době už probíhá rozpoznávání znaků vesměs v digitální rovině, nikoli opticko-mechanickou cestou.



Obrázek 1.:Reading Machine

Jako první si aplikaci tohoto systému zakoupila firma Readers Digest, která pro ni našla využití v oddělení pro předplatné. Významným odběratelem byla také kalifornská společnost Standard Oil Company, která OCR začala využívat pro čtení obtisknutých čísel kreditních karet na účtech.
V první polovině šedesátých let minulého století, bylo jisté, že se proces OCR musí standardizovat. Základem bylo použití tzv. OCR písma, které bylo lépe čitelné systémy OCR. Byla používána tiskárna Drum RCA, která tiskla dokumenty právě písmem vhodným pro OCR. Od roku 1965 se začalo OCR používat u USPS (The United States Postal Service). V Evropě se OCR začalo poprvé používat v roce 1965 ve Velké Británii u GPO (General Post Office). Roku 1971 začala OCR používat také kanadská pošta. Nejprve se používalo ke čtení jména a adresy adresáta, později se už používalo pouze k třídění dopisů podle PSČ.

V roce 1974 založil Ray Kurzweil společnost Kurzweil Computer Products a vedl vývoj prvního počítačového OCR programu. Rozhodl, že nejlepší program, který bude využívat technologii OCR, bude program pro zrakově postižené. Ti by díky takovému programu dokázali porozumět tištěnému textu, protože by jim tento text počítač sám předčítal. Nicméně toto řešení vyžadovalo vynález dvou technologií. CCD skener a syntetizátor řeči pro počítače. 13.ledna 1976 byl dokončen program, který se jmenoval Kurzweil Reading Machine. Vůbec prvním zákazníkem firmy Kurzweil Computer Products, který si Kurzweil Reading Machine osobně koupil, byl hudebník Stevie Wonder.

V roce 1978 začala firma Kurzweil Computer Products prodávat komerční verzi svého OCR programu. Firma LexisNexis používala program pro převod jejich papírových dokumentů do jejich online databáze. O dva roky později prodává Kurzweil svou společnost firmě Xerox, která chtěla dále rozšiřovat problematiku převodu tištěného textu do elektronické podoby. Kurzweil Computer Products se stává dceřinou společností firmy Xerox.

V dnešní době jsou již OCR programy rozšířené i mezi běžné uživatele. Jedná se zpravidla o software, který analyzuje obrázek a snaží se v něm rozpoznat text, který pak v co nejvěrnější podobě uloží do editovatelného formátu (TXT, RTF, DOC apod.). Pokročilejší aplikace umí uložit i tabulky, zachovat řezy písma nebo vložit obrázky.

Právě různá písma a grafické prvky rozpoznávání velice znesnadňují. Další problémy mohou nastat, pokud je text nakloněn, případně nekvalitně reprodukován či nasnímán. Kromě toho jsou v některých písmech skupiny písmen, která mají tendenci tvořit nerozlučné dvojice, čímž opět znesnadňují rozpoznání. Známou takovou dvojicí je "r n", které velice připomíná písmeno "m".

Na scéně se objevují tzv. „Inteligentní“ OCR systémy, které vylepšují standardní OCR systémy o schopnost se učit a tak se postupně zlepšovat. Toto je ale podmíněno o nutnost označení, zda bylo rozpoznání kvalitní, případně kde bylo chybné. Těmto systémům se pak říká ICR systémy.[1]

V současnosti jsou nejznámějšími produkty v oblasti OCR ABBYY FineReader, MS Office Document Imaging a OmniPage, který byl vyvíjen firmou Xerox a později prodán.

1.2 Současné nejrozšířenější OCR

V dnešní době patří mezi nejrozšířenější software, řešící problematiku převodu tištěného textu do elektronické podoby, ABBYY FineReader, který dosahuje výborné výsledky ve zmíněné problematice. Tento program je nyní k dispozici ve své 10. verzi, takže je dostatečně znát několikaletý vývoj problematiky OCR. V běžném životě se však nejčastěji setkáváme s jiným OCR nástrojem, než je rozpoznávání samotného tištěného textu. Vůbec nejrozšířenějším nástrojem, který rozpoznává tištěné znaky a který si většina z nás ani neuvědomuje, že do této skupiny patří, je nástroj k rozpoznávání určitých značek a kódů. Tímto nástrojem je nám všem velice známá čtečka čárových kódů u pokladny v obchodě.

1.2.1 Čárový kód

Čárový kód je prostředek pro automatizovaný sběr dat. Je tisknut pouze černobílým tiskem a to vytištěnými pruhy definované šířky, které jsou čitelné pomocí speciálního přístroje. Tomuto přístroji se říká skener, častěji však čtečka čárového kódu pro jednorozměrné kódy, či speciální skener pro jednorozměrné nebo dvourozměrné kódy. Patent na čárový kód byl poprvé udělen v roce 1949. Tehdy se v žádném případě nepočítalo s tím, že se čárový kód postupem času tak rapidně rozšíří po celém světě. Čárové kódy se dělí do různých skupin podle způsobu kódování těchto kódů. V současné době je definováno přibližně 200 různých standardů čárových kódů. [2]

1.2.2 Konstrukce čárového kódu

Každý čárový kód je tvořen sekvencí čar a mezer s definovanou šířkou. Ty jsou při čtení transformovány podle své sytosti na posloupnost elektrických impulsů různé šířky a porovnávány s tabulkou přípustných kombinací. Pokud je posloupnost v tabulce nalezena, je prohlášena za odpovídající znakový řetězec. Nositelem informace je nejenom tištěná čára, ale i mezera mezi jednotlivými dílčími čarami. Krajní skupiny čar mají specifický význam - slouží jako synchronizační pro čtecí zařízení, které podle nich generuje signál Start/Stop. Technická specifikace pak vyžaduje ochranné světlé pásmo bez potisku před a za synchronizačními čarami.

1.2.3 Některé čárové kódy

Následuje výpis některých nejrozšířenějších jednorozměrných a dvourozměrných čárových kódů. U každé skupiny čárových kódů je přidán pro ilustraci obrázek s ukázkou

Kódy typu EAN

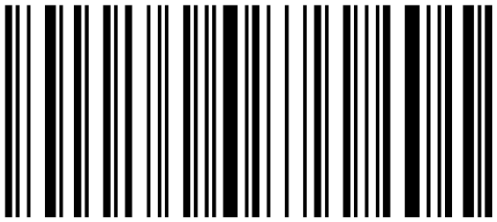
Kódy typu EAN (European Article Number) patří do skupiny jednorozměrných čárových kódů a jsou vůbec nejrozšířenějšími čárovými kódy. Přesněji to jsou kódy typu EAN-13. Kódy typu EAN-13 jsou používány po celém světě k označování jednotlivých druhů zboží. Upravená podoba tohoto kódu například umí uchovávat ISBN kódy knížek nebo ISSN kódy časopisů a jiných periodik. Z kódu EAN-13 lze také zjistit zemi původu výrobce nebo způsob užití daného zboží. Méně jsou používány kódy EAN-8, které jsou vyhrazeny a používány pro menší položky, na které je problém umístit 13místný kód, jako jsou třeba cukrovinky. U kódů typu EAN-13 se právě 13 čísel kóduje do 4 různých skupin. První skupinou jsou systémové číslice. Tato skupina obsahuje první dvě nebo tři číslice, které obvykle identifikují zemi, kde je zaregistrovaný výrobce označeného výrobku. Druhá skupina obsahuje další čtyři nebo pět číslic a slouží k identifikaci výrobce. Třetí skupinu, která zastupuje kód výrobku, tvoří pět číslic. Poslední skupinou je tzv. „kontrolní číslice“.

Obrázek 2.: Kódy typu EAN

Code 128

Dalším typem jednorozměrného čárového kódy jsou kódy s názvem Code 128. Již z názvu je patrné, že tento kód dokáže zakódovat celkem 128 znaků, a to spodní polovinu ASCII. Jako jeden z mála znaků umí rozlišovat a zachovat velikost písmen v kódu. Každý znak kódu se skládá ze tří čar a

tří mezer definované šířky. Kódy tohoto typu se používají v logistice nebo například k označování patentů.



Obrázek 3.: Kód ze skupiny Code 128

Data Matrix

Příkladem dvourozměrného čárového kódu je kód typu Data Matrix. Tento kód je čtvercový s velikostmi od 8×8 do 144×144 bodů a jeho předností je to, že umí zakódovat celou ASCII tabulku znaků. Pro větší vstupní data se dělí na menší části, z nichž každý obsahuje tzv. „tichou zónu“. Tato zóna představuje levý a dolní černý okraj a nenese žádné informace. Data Matrix obsahuje algoritmy korekce chyb Reed-Solomon. Užívá se v některých průmyslových úsecích. Například jsou jimi označována sériová čísla některých počítačových komponent. Data Matrix byl vyvinut v říjnu roku 2005.

Obrázek 4.: Data Matrix

Kruhový kód

Posledním příkladem čárových kódů je tzv. „kruhový kód“. Tento kód již není standardizován, jako všechny výše zmíněné typy čárových kódů. Je zobrazen ne jako sled čar, ale jako spojení těchto čar do soustředných kružnic. Výhodou je to, že poloha čtečky kódů vůči kódu může být libovolná, nevýhodou je pak ale větší nárok na místo pro záznam kódu.

Obrázek 5.: Kruhový kód

1.3 Současný stav

Pokud máme k dispozici kvalitní zdrojový oskenovaný nebo vytištěný text, je rozpoznání latinky u tištěného textu v současné době považováno za aplikačně vyřešitelný problém. V takovém případě se přesnost rozpoznání dostává až na hranici 99%. Opravdu přesné rozpoznání je dosáhnuto pouze s lidským zásahem. Rozpoznání ostatních problémů, jako jsou například ručně psaný text, zahnutý text a text psaný v jazyce s velkým počtem znaků, jako je například čínština, je stále předmětem aktuálního výzkumu.

Přesnost rozpoznání lze ovlivnit různými způsoby podle kvality vstupního obrazu. Dnešní OCR software většinou používají tzv. „slovník chyb“, pomocí kterého se právě nalezené chyby eliminují až na hodnotu kolem 1%. Pokud ale software takový slovník nemá, může se procentuální úspěšnost špatného rozpoznání dostat k hranici 5% nebo i více v případě, že je software založený na nesprávném rozpoznávání.

OCR je tzv. „off-line“ metodou rozpoznávání znaků, kdy software rozpoznává předem určený text. Naopak skupina „on-line“ rozpoznávání obsahuje software, dynamicky rozpoznávající text při psaní. Dynamické rozpoznávání se používá například u tabletů. Této skupině se také někdy říká dynamické rozpoznávání znaků nebo také inteligentní rozpoznávání znaků (ICR).

Právě software rozpoznávající ručně psaný text se stal během několika let komerčně úspěšným a to hlavně díky zařízením PDA, kde se tohoto softwaru bohatě využívá. Používané algoritmy pracují s tím, že pořadí, směr a rychlost psaní nejsou předem známy. Někdy se musí sám uživatel přizpůsobit možnostem poskytovaných daným softwarem. V takovém případě se přesnost rozpoznání pohybuje v rozmezí od 80% do 90%, kterou ale uživatel může patřičně vylepšit svým zásahem. Ovšem při rozpoznávání tištěného textu existují stále velké nedostatky hlavně kvůli nutnosti opravy různých chyb po rozpoznání.

Rozpoznání prohnutého textu je v současné době jedním z největších problémů celého OCR. Opravdu kvalitní rozpoznání prohnutého textu nebude pravděpodobně nikdy možné bez pomoci určitých gramatických nebo kontextových úprav. Například rozpoznání celého slova je v situaci, kdy má software možnost pracovat se slovníkem, podstatně jednodušší než rozpoznání jediného znaku. Znalost gramatiky rozpoznávaného jazyku může v některých případech pomoci rozlišit, zda je rozpoznávané slovo slovesem nebo podstatným jménem. Ale právě tvary některých prohnutých písem nedokážou být tak dobře rozpoznané, aby bylo dosaženo použitelného výsledku.

V současnosti se rozšiřuje metoda rozpoznávání obtížně rozpoznatelného textu za pomocí uživatele. Jako jsou například různé kontrolní body při registraci do různých internetových systémů. Takové metodě se již běžně říká reCAPTCHA. [3]

1.4 Problematika řešená v mé práci

Tak jak jsou v určitých oblastech stanoveny meze, kterých je možno dosáhnout, tak je také u problematiky OCR stanovena mez. Touto mezí je problematika rozpoznání ručně psaného textu. V současnosti je prakticky nemožné uvažovat o vývoji nástroje pro rozpoznávání ručně psaného textu. Hlavním důvodem je rukopis každého z nás. Právě proto jsem se v této práci zaměřil na problematiku rozpoznání tištěného textu. Ať už jde o oskenované stránky časopisů nebo o staré spisy psané psacím stojem. Ovšem jeden problém je u komerčních programů poměrně nedořešen. Tímto problémem je deformovaný text (text na kružnici, na minci, prohnutý text, atd.). Proto jsem do své práce zařadil také problematiku rozpoznávání různě deformovaných textů. Vůbec základním kamenem pro kvalitní OCR software je problém kvalitního předzpracování vstupního obrazu, pro co nejkvalitnější výsledek při samotném OCR. Proto je důležité mít vstupní obraz pro fázi samotného převodu tištěného textu ve formátu dvoubarevné podoby (tj. černobílý obraz). Vstupní obraz pak musí splňovat určité předpoklady. Ať už jde o to, mít černý text na bílém pozadí, tak snahu o co nejkvalitnější „vyčištění“ vstupního obrazu od nežádoucích faktorů. Mezi tyto faktory patří šum, způsobený při nekvalitním skenování, světlý text na tmavém pozadí nebo text v obrázku. V této práci bude tedy kladen velký důraz na samotné předzpracování vstupního obrazu a snahu tento text co nejlépe připravit pro fázi OCR. Vstupní obraz pak bude uložen v černobílém formátu, který je nejvhodnějším pro potřebný převod.

1.5 Cíle práce

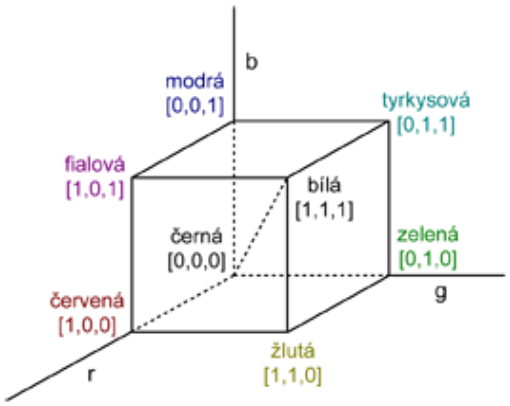
Cílem celé práce bylo implementace nástroje pro co nejkvalitnější přípravu vstupního obrazu při použití v problematice OCR. Samozřejmostí je pak dále možnost volby relevantních oblastí pro další použití, jako je například speciální úprava obrazu v této oblasti. Snaha o vyřešení problematiky textu na kružnici a prohnutého textu a zařazení této problematiky do nástroje s možností tento text dále převést do elektronické podoby. V závěru pak problematika hromadného zpracování tištěného textu s co nejmenším zásahem uživatele.

2. Problematika zpracování tištěného textu

Následující problematika obsahuje popis možných řešení při zpracování tištěného textu. Popis je univerzální a tedy řešení nejrůznějších problémů lze implementovat v jakémkoliv prostředí. V určitých případech může dojít k drobným úpravám. Hlavním problémem samotného zpracování tištěného textu je řešení vad a nedokonalosti vstupního obrazu. Právě této problematice je věnována velká část této kapitoly.

2.1 Barevný model RGB

Model RGB se nejčastěji používá pro snímání barevného obrazu a zobrazování na monitoru. Barevný obraz dle tohoto standardizovaného modelu je složen ze tří barev. Červené (red), zelené (green) a modré (blue). Jednotlivým složkám byly přiřazeny vlnové délky. Red = 700nm, green = 546,1nm a blue = 435,8nm. Z technického hlediska je pak každá barva reprezentována určitým počtem bitů. Obvykle to bývá 8bitů pro každou barvu. Pokud pak složíme jednotlivé barevné složky RGB v patřičném poměru, získáme výslednou barvu. Barevný prostor modelu RGB se označuje jako aditivní skládání barev. To znamená, že čím vyšší je hodnota, tím je barva světlejší. Model si ukážeme na jednotkové krychli, kde počátek souřadnic odpovídá černé barvě (R = 0, G = 0, B = 0), vrchol souřadnic pak odpovídá barvě bílé (R = 1, G = 1, B = 1). Barvy červená R, zelená G a modrá B leží v protilehlých rozích a vektor (R, G, B) nakonec určuje výslednou barvu, které odpovídá jeden bod krychle.



Obrázek 6.: Jednotková krychle barevného modelu RGB

2.2 Řešení vad a nedokonalosti vstupních materiálů

Před možností kvalitního a pokud možno minimálně chybového převodu tištěného textu do elektronické podoby, je potřeba vstupní materiál co nejkvalitněji připravit pro finální fázi celého převodu. Tato příprava spočívá v analýze a následném řešení úpravy nedokonalého vstupního obrazu. Mezi takové úpravy patří ať už jednoduché otáčení obrazu nebo složitější filtrování šumu a hranová detekce obrazu.

2.2.1 Převod barevného obrazu RGB na odstíny šedé

Barevný obraz můžeme převést na obraz v odstínech šedé, kdy jsou tyto odstíny vyjádřeny jako hodnoty jasu. Barevný obraz se v tomto případě převádí do odstínů šedé z jednoho hlavního důvodu. V převedeném obraze se už totiž nemusí zkoumat barevné složky separátně, takže se pak může takový obraz dále jednoduše zpracovávat. V našem případě to jsou například různé hranové detektory, které pracují s obrazem v odstínech šedé. Lidské oko je různě citlivé na jednotlivé složky barev RGB, a tudíž není možné převod provádět pouhým zprůměrováním jednotlivých složek na bodu v barevném obraze. Pro převod se používá takzvaných váhových koeficientů. [4]

$$f = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B$$

Vzorec 1.: Váhový koeficient pro převod RGB do odstínu šedé

Ve výše zmíněném vzorci představují R, G a B úroveň jasů jednotlivých složek daného bodu v barevném obraze a f představuje výslednou úroveň jasu v obraze převedeném na odstíny šedé.

Obrázek 6.: Barevný obrázek

Obrázek 7.: Obrázek převedený do odstínu šedé

2.2.2 Geometrické transformace

Při úpravách vstupního obrazu se může použít tzv. „geometrické transformace“. Tyto transformace se používají k úpravě zdeformovaných tvarů nebo naopak k záměrnému zdeformování obrazu. Mezi nejjednodušší a taky nejčastější afinní transformace patří posunutí, otočení, změna měřítka a zkosení. V případě jako je odstranění nedokonalosti vad vstupního obrazu se nejčastěji používá transformace otočení.

$$b_x = a_x \cos\varphi - a_y \sin\varphi$$
$$b_y = a_x \sin\varphi + a_y \cos\varphi$$

Vzorec 2.: Transformace otočení bodu okolo počátku souřadnic o úhel φ

Hodnoty a_x, a_y značí souřadnice původního bodu a b_x, b_y značí souřadnice bodu nového. Další často používanou transformací je pak transformace změna měřítka, která je více známá pod názvem *zoom*.

$$b_x = s_x a_x$$
$$b_y = s_y a_y$$

Vzorec 3.: Transformace změna měřítka

Transformace se ale nejlépe provádějí v homogenních souřadnicích, kde se může s výhodou použít maticový zápis. Souřadnice bodu, který se má transformovat, jsou zapsány jako vektor v homogenních souřadnicích $\bar{a} = [a_x, a_y, 1]$. Souřadnice transformovaného bodu jsou pak v homogenních souřadnicích vyjádřeny $\bar{b} = [b_x, b_y, 1]$ a M je transformační matice.

Početní vztah pro souřadnice transformovaného bodu je pak:

$$\bar{b} = \bar{a} \cdot M$$

Vzorec 4.: Vztah pro výpočet transformovaného bodu

Stejně jako lze vyjádřit souřadnice bodu v homogenní soustavě souřadnic, jde také vyjádřit samotnou transformaci.

$$[b_x, b_y, 1] = [a_x, a_y, 1] \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vzorec 5.: Transformace otočení okolo počátku souřadnic o úhel φ

Transformace lze velice jednoduše skládat. Výhodou maticového zápisu je pak to, že pokud bychom chtěli složit více transformací, stačí jednotlivé transformační matice vynásobit. Touto výslednou maticí pak vynásobíme všechny požadované body v patřičné oblasti. Výhodou takového skládání je to, že se ušetří spousta operací, pro provádění jednotlivých kroků. Když si vezmeme příklad jako obrázek o velikosti 100 x 100 bodů a budeme chtít provést otočení tohoto obrázku kolem jiného bodu než je počátek. Pak nebude třeba provádět transformaci a počítat deset tisíc bodů pro posunutí do počátku, deset tisíc bodů pro otočení a následně deset tisíc bodů pro posunutí zpět. Místo toho spočítáme výslednou matici složené transformace, která je dána součinem matice pro jednotlivé transformace. [4]

Obrázek 8.: Obrázek po transformaci otočení o 45°

2.2.3 Jasové transformace

Při bodové jasové transformaci dochází ke změně jasové stupnice. Změna intenzity obrazového bodu nezávisí na poloze bodu v obrazu.

Inverze barev

Tato metoda patří do skupiny jasových transformací. Při invertování obrazu se hodnota jasu každého obrazového bodu v kanálech převede na inverzní hodnotu na stupnici barevných hodnot s 256 kroky. Například obrazový bod s hodnotou 255 v pozitivním obraze se změní na 0 a obrazový bod s hodnotou 5 se změní na 250. Operací invertování se nezmění hodnota RGB kanálu v určitém bodě. Změní se pouze hodnota obsahu určité barevné složky v bodě.

Obrázek 9.: Obrázek po provedení invertování barev

Gama korekce

Gama korekce se využívá nejen pro úpravy jasu na obrazovce, ale také pro úpravu špatné expozice. Dosahuje dobrých výsledků v problémech jako je například problém špatného nasvícení určité části obrazu nebo tzv. „ukrytí“ určité části obrazu v příliš tmavé nebo příliš světlé části obrazu. Příkladem je korekce snímků za účelem zviditelnění některých tmavých částí v protisvětle. Gama korekce se původně využívala kvůli nelineární závislosti televizních obrazovek. Klasické obrazovky se žhavenou katodou CRT totiž nezobrazovaly vstupní napětí lineárně, ale výsledný průběh intenzity jasu odpovídal přibližně křivce $g = c \cdot (f + \varepsilon)^{\gamma}$, kde γ je konstanta a ε je tzv. „offset“, tedy posun vzhledem k počátku [4]. Vliv zmíněného offsetu ε se většinou zanedbává a výsledný vztah je interpretován jako:

$$g = c \cdot f^{\gamma}$$

Vzorec 6.: Matematický vztah pro gama korekci

Obrázek 10.: Příklad obrázku před a po gama korekci

Logaritmická transformace

Při některých výpočtech týkajících se zpracování obrazu, vycházejí výsledné hodnoty poměru minimální/maximální hodnota v příliš velkém rozsahu. Ve zpracování obrazu se někdy používají transformace, kdy rozsah hodnot obsažených ve výsledku je běžně od 0 do 10^6. Takový rozsah hodnot by ale nebylo možné rozumně zobrazit na monitoru. Proto je nutné tento interval hodnot tzv. „zkomprimovat“ tak, aby byl dobře viditelný na obrazovce. Tento proces je právě úkolem logaritmické transformace.

Obrázek 11.: Příklad obrázku před a po logaritmické transformaci

Roztažení kontrastu

Cílem této transformace je zvýšení kontrastu obrazu. Tato metoda se převážně používá, pokud je rozdíl mezi různými úrovněmi jasu v obraze příliš malý, tím se stává obrázek nekontrastním a pro nás zajímavá informace se nachází v příliš úzkém pásmu jasových úrovní. Tento problém nemusí být způsoben pouze vlivem nevhodného osvětlení. Pokud si vezmeme jako příklad zašlý nápis na archeologických vykopávkách nebo staré razítko na dopisní obálce. Zde jsou jasové hodnoty objektu, který nás zajímá, pouze v rozmezí několika procent. V takovém případě pomůže právě roztažení kontrastu. Tato metoda se dále hodí při zpracovávání starých a zašlých dokumentů, které byly psané na psacím stroji. Zde se naprosto ideálně hodí její aplikace.

Roztažení kontrastu je možné realizovat jako po částech lineární transformaci. Pokud budeme předpokládat, že minimální hodnota jasu vyskytující se v obraze je f_1 a maximální hodnota pak bude f_2. Hodnoty z tohoto intervalu bude potřeba převést na celý zobrazitelný rozsah hodnot. $f \in \langle 0, 1 \rangle$. Transformaci roztažení kontrastu pak realizujeme tak, že od dané hodnoty jasu f odečteme minimální hodnotu nalezenou v obraze a vynásobíme koeficientem roztažení $1/(f_2 - f_1)$. [4]

$$g = \frac{f - f_1}{f_2 - f_1}$$

Vzorec 7.: Matematický vztah pro roztažení kontrastu

Obrázek 12.: Příklad obrázku před a po roztažení kontrastu

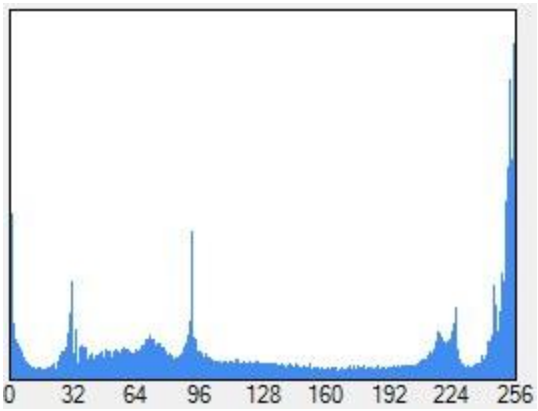
2.2.4 Histogram

Histogram představuje různé rozložení jasových úrovní v obraze. Tzn. četnosti jednotlivých úrovní jasu. Histogram digitálního obrázku s úrovní jasu $[0, \dots, L-1]$ je vyjádřen jako diskrétní funkce:

$$h_{(f_k)} = n_k$$

Vzorec 8.: Diskrétní funkce histogramu

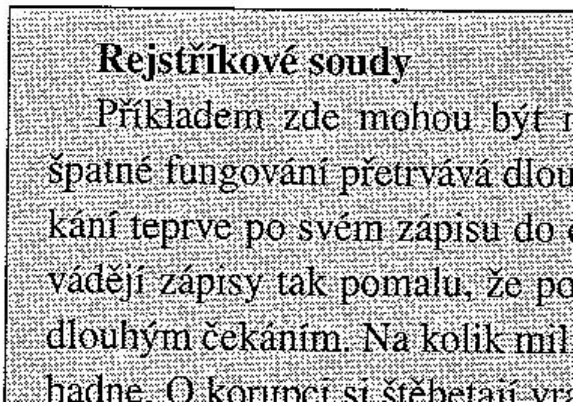
kde f_k je k - tá úroveň jasu. Takže $k \in \langle 0, (L-1) \rangle$ a n_k je počet pixelů v obraze s úrovní jasu f_k. Tímto jsme získali potřebné údaje pro vytvoření histogramu. Histogram se většinou uživateli zobrazuje jako graf, kde osa x představuje f_k a je rozdělena na L hodnot $([0, \dots, L-1])$ a osa y představuje četnost n_k. Informace, které jsou získané z histogramu, bývají nejčastěji využity při operacích s obrazem. Ať už to je vylepšení vzhledu obrazu nebo prahování, o kterém se zmíním později. V barevném obrazu bude pro každou barevnou složku odpovídat jeden histogram. Každý takový histogram se může výrazně lišit. Takže pro jeden obraz budou existovat celkem 4 histogramy. 3 histogramy pro jednotlivé barevné složky a poslední histogram představuje celkovou jasovou složku. Právě histogram představující jasovou složku se nejčastěji využívá při prahování šedotónového obrazu.



Obrázek 13.: Příklad grafického zobrazení histogramu

2.2.5 Vyhlazování obrazu

Zdrojový obraz většinou ovlivňují některé nežádoucí jevy. Mezi takové jevy patří například tzv. „šum“. Šum se může projevovat jako zrnění obrazu. Většinou lze tyto nežádoucí jevy v obraze potlačit vyhlazením obrazu. Jevů, které způsobují špatné převody obrazu do vstupního obrazu pro samotnou fázi OCR, je více druhů a vždy je třeba vhodně zvážit efektivitu metody, která dokáže nežádoucí jevy potlačit.



Obrázek 14.: Příklad šumu v obraze

Vyhlazování průměrováním

Vůbec nejjednodušší metodou pro vyhlazování obrazu je metoda vyhlazování průměrováním. U této metody se jedná o vyhlazování obrazu pomocí konvoluce, která se používá právě při filtraci obrazu, za použití masky. Tato maska zajišťuje, že výsledkem konvoluce je průměr hodnot jasu ze sousedních bodů v obraze. Pokud zvážíme, že velikost masky bude 3 x 3, pak vypadá matematické vyjádření metody vyhlazování průměrováním takto:

$$g(x,y) = \frac{1}{9}\sum_{s=-1}^{1}\sum_{t=-1}^{1} f(x-s, y-t)$$

Vzorec 9.: Matematické vyjádření vyhlazování průměrováním

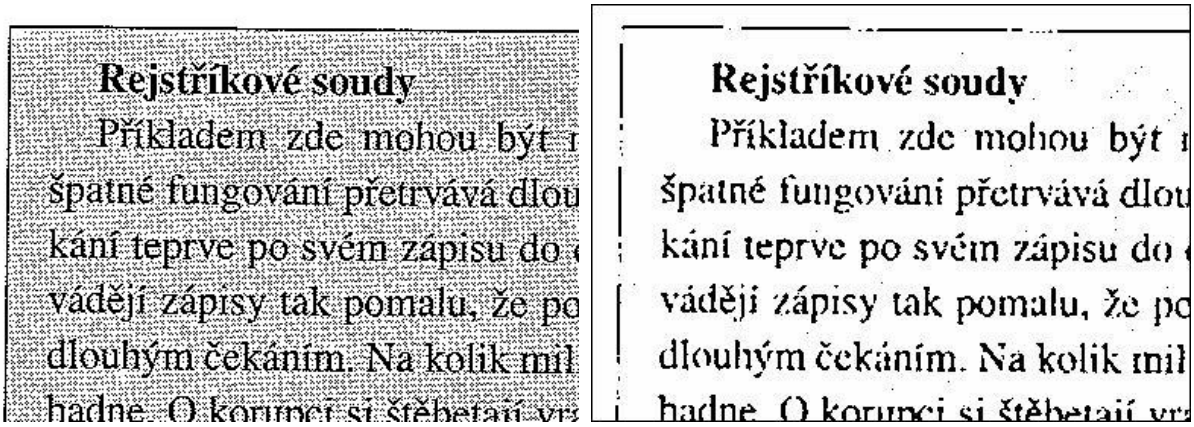
Konvoluční maska pro použitou metodu ve velikosti 3 x 3 vypadá následovně:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

Vzorec 10.: Konvoluční maska (matice) pro vyhlazování průměrováním

Mediánová filtrace

Mediánová filtrace vykazuje v některých příkladech opravdu kvalitní výsledky. Tento filtr se také nejčastěji používá při odstraňování nežádoucího šumu ve špatně oskenovaném obraze. Samotný medián je prostřední prvek v uspořádané posloupnosti hodnot. Při lichém počtu prvků n je to prvek $f_{med} = f_{(n+1)/2}$. Pokud je ale počet prvků v posloupnosti sudý, je medián vypočítán jako aritmetický průměr obou postranních hodnot $f_{med} = \left(f_{n/2} + f_{(n+1)/2}\right)/2$. Tato filtrace je příkladem nelineární filtrace obrazu. Princip filtrace spočívá v posouvání masky po obraze a výběru mediánu z hodnot ležících v obraze pod zmíněnou maskou. Jak bylo výše zmíněno, při použití vyhlazování průměrováním metodou s konvolucí, tak u této metody se konvoluce nepoužívá. Většinou se používá maska o velikosti 3 x 3 nebo 5 x 5. V těchto případech se vybere z masky pátý respektive třináctý největší prvek, který bude představovat právě hodnotu mediánu. [4]



Obrázek 15.: Příklad obrazu před a po použití metody mediánová filtrace

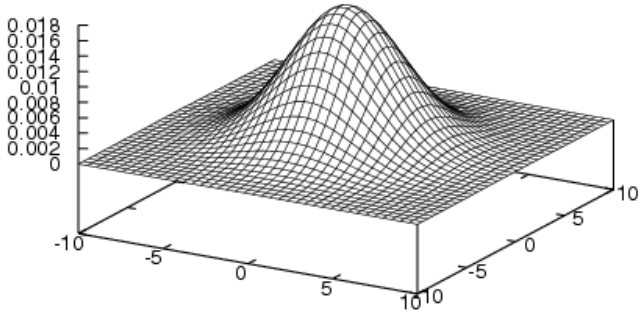
Gaussovo vyhlazování

Poslední zmíněnou metodou pro vyhlazování obrazu je Gaussovo vyhlazování. U této metody se počítá s tím, že koeficienty blíže středu masky mají vyšší váhu a odpovídají hodnotám na Gaussově křivce. Gaussovo vyhlazování patří mezi konvoluční filtry. Tyto filtry se aplikují postupně na všechny body obrazu a používají váženého součtu hodnot z okolí daného bodu. Pro konvoluční filtry je typické, že se ve výsledku mohou objevit hodnoty, které v původním obrázku nejsou. Gaussovo rozdělení, kde střední hodnota je v bodě $(0, 0)$ je dáno vztahem:

$$G(x, y) = \frac{1}{2\pi\sigma^2} exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

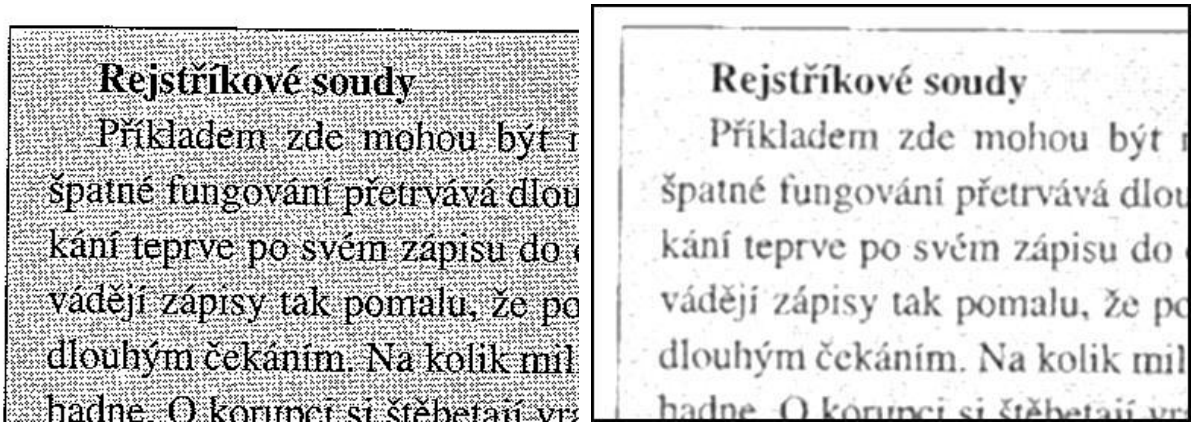
Vzorec 11.: Vztah pro Gaussovo rozdělení

Na následujícím obrázku je znázorněn tvar hustoty pravděpodobnosti pro dvourozměrný náhodný vektor s Gaussovým rozdělením. [4]



Obrázek 16.: Příklad Gaussova rozdělení

Funkce hustoty je definována pro všechny body v rovině v intervalu $(-\infty, \infty)$. Při implementaci této metody v diskrétním případě je ale nutné omezit velikost masky.



Obrázek 17.: Příklad obrazu před a po použití metody Gaussovo vyhlazování

2.2.5 Detekce hran

Detekce hran se ve fázi předzpracování vstupního obrazu pro převod tištěného textu do elektronické podoby používá převážně pro získání textu z barevného obrazce, kdy dosahuje opravdu kvalitních výsledků. Hrany v obraze totiž odpovídají prudkým změnám hodnot jasu. Základní myšlenkou detekce hran je najít místa v obraze, kde se jas výrazně mění. Tyto změny se dají detekovat pomocí prvních a druhých derivací intenzity jasu. Kritérium pro detekci těchto změn je velikost první derivace intenzity jasu, velikost druhé derivace intenzity jasu nebo také detekce změny znaménka derivace.

První derivace

Při použití první derivace, jak metody pro detekci hran, se výsledku také říká gradient. Tato derivace může být definována ve dvou směrech, a to buď podle osy x nebo podle osy y. Matematické vyjádření gradientu funkce dvou proměnných $f(x, y)$ se definuje jako vektor:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Vzorec 12.: Matematické vyjádření gradientu funkce dvou proměnných

Gradient, hlavně z důvodu kolmosti na hranu, lze pak dále využít jako informaci pro hledání hran. Odhad derivací G_x a G_y v diskrétním obraze se dá vypočítat rozdílem dvou sousedních hodnot:

$$G_x \approx f(x+1, y) - f(x, y)$$

Vzorec 13.: Rozdíl dvou sousedních hodnot při odhadu derivací

V případě dvojnásobné velikosti vektoru se může použít také symetrická varianta, ve které se místo $f(x, y)$ použije $f(x-1, y)$.

Operátorů, pracujících pří detekci hran na principu gradientu je několik. Vůbec nejjednodušším je Robertsův operátor, který je založen na konvoluci. U této metody se používá konvoluční maska o velikosti 2 x 2.

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$

Vzorec 14.: Konvoluční maska pro výpočet Robertsova operátoru

Masky o velikosti 2 x 2 už nebývají tak moc obvyklé. Mnohem častěji se používají masky o velikosti 3 x 3. Při této velikosti masky se provádí rozdíl hodnot $f(x+1, y) - f(x-1, y)$, respektive $f(x, y+1) - f(x, y-1)$. Nejčastějšími operátory, které využívají právě masku velikosti 3 x 3 jsou operátory Prewittové a Sobelův.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Vzorec 15.: Masky pro směr x a y k výpočtu operátoru Prewittové

Dalším operátorem je již výše zmíněný Sobelův operátor. U tohoto operátoru se volí váhy tak, aby byly zdůrazněny hodnoty blíže středu masky. Jde také masky rotovat o $45°$ a počítat tak směrové derivace nejen ve směru x a y.

V praxi dosahuje Sobelův operátor lepších výsledků při detekci textu v barevném poli a to taky díky tomu, že nebývá příliš citlivý na šum. Naopak operátor Prewittové ukazuje použitelnější výsledek při detekci rytého textu jako je například text na minci nebo text na pečeti.

$$\begin{bmatrix} -1 & 0 & 1 \\ 2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

Vzorec 16.: Masky k výpočtu Sobelova operátoru s rotací o 45°

Poslední zmíněnou metodou pro detekci hran v obraze, která využívá první derivaci, je Cannyho hranový detektor. Tento algoritmus bývá někdy zvýhodňován před ostatními metodami. Hlavním přínosem jsou opravdu kvalitní výsledky. To je ovšem připsáno na úkor vysoké časové náročnosti. Při zpracování jediného obrazu se tyto nároky můžou zanedbat, ale při hromadném zpracování textu by časová kvóta na detekci hran postupně narůstala. Právě z tohoto důvodu se Cannyho hranový detektor používá u problematiky hromadného zpracování tištěného textu jen zřídka.

Obrázek 18.: Obrázek před a po použití operátoru Prewittové ve směru osy y

Obrázek 19.: Obrázek před a po použití Sobelova operátoru při rotaci 45°

Jak je již z obrázků patrné, tak Sobelův operátor opravdu dosahuje kvalitnějších výsledků při detekci textu v barevném poli. Ovšem požadovaný výsledek pro fázi OCR by se musel ještě lépe upravit.

Druhá derivace

U druhé derivace se už přihlíží k tzv. „rychlosti změny jasu“. Někdy se této metodě také říká hledání změny ve změně. Její použití je vhodné zejména na obraz s velkým počtem strmých nebo izolovaných hran. Metoda lze dále použít pro detekci izolovaných bodů. Nevýhodou je pak ale zvýrazňování šumu v obraze. Druhou derivaci ve směru x lze v diskrétním obraze počítat jako rozdíl rozdílů hodnot jasu ležících vedle sebe:

$$\partial^2 f(x,y)/\partial x^2 \approx [f(x+1,y) + f(x-1,y) - 2f(x,y)]$$

Vzorec 17.: Matematické vyjádření druhé derivace ve směru x

Obdobně lze druhou derivace počítat ve směru y, kdy se pouze přičte/odečte 1 od proměnné y. Lepších výsledků, ve srovnání metody ve směru x a y pak dosahuje metoda ve směru x.

K detekci izolovaných bodů lze použít tzv. „Laplaceův operátor“:

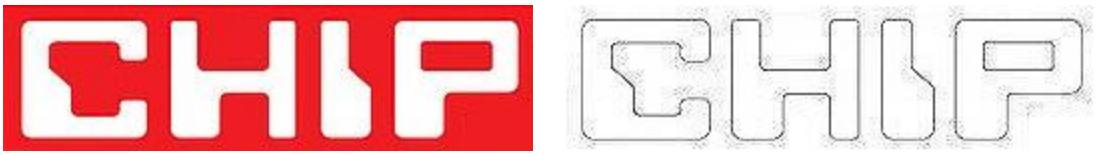
$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

Vzorec 18.: Matematické vyjádření Laplaceova operátoru na funkci f v bodě (x,y)

Pro Laplaceův operátor se používají konvoluční masky ve dvou variantách. A to ve variantě pozitivní nebo negativní podle požadovaného výsledku obrazu.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Vzorec 19.: Konvoluční masky pro Laplaceův operátor pro pozitivní a negativní variantu



Obrázek 20.: Obrázek před a po použití Laplaceova operátoru ve směru x v negativní variantě

Laplacian of Gaussian

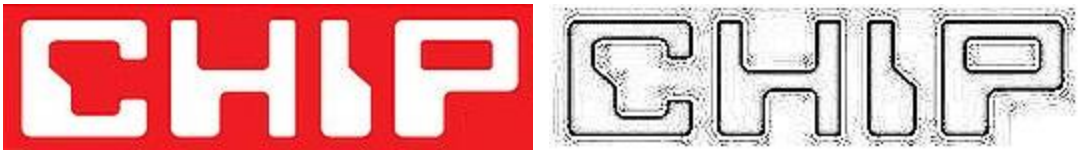
Samotný Laplaceův operátor je příliš citlivý na šum v obraze a navíc jeho použití může způsobit nalezení dvojitých hran bez možností určení směru hrany. I přes uvedené nevýhody lze Laplaceův operátor úspěšně použít v kombinaci s jinou metodou. Samotné zvýrazňování hran lze dobře provádět pomocí kombinace Gaussova a Laplaceova operátoru. Konvolucí obrazu a Gaussova operátoru bychom dosáhli žádaného rozmázaní. Hodnoty šumu by potom nezpůsobovaly tak velkou odezvu na konvoluci obrazu a Laplaceova operátoru. Samozřejmě není podmínkou provádět nejprve konvoluci s Gaussovým operátorem a poté znovu konvoluci s Laplaceovým operátorem, obě metody lze spojit do jediné. Žádaný výraz pak vypadá takto:

$$\nabla^2 h(r) = -\frac{r^2 - 2\sigma^2}{2\pi\sigma^6} e^{-\frac{r^2}{2\sigma^2}}$$

Vzorec 20.: Matematické vyjádření metody Laplacian of Gaussian, kde $r^2 = x^2 + y^2$

Tato metoda dosahuje podobných výsledků jako výše zmíněný operátor Prewittové. V některých situacích je ale lepší použít právě Laplacian of Gaussian. Převážně při špatně nasvětleném vstupním obraze, protože metoda Laplacian of Gaussian dokáže díky požadovanému rozmazání Gaussovým

operátorem odstranit nežádoucí šum, který vzniká při nevhodném nasvětlení. Zejména pokud byl použit nekvalitní skener nebo fotoaparát. [4]



Obrázek 21.: Obrázek před a po použití operace Laplacian of Gaussian

2.2.6 Prahování

Tato metoda patří do skupiny úpravy obrazu, které se říká segmentace obrazu. Segmentace znamená rozdělení obrazu na části, které nás zajímají a ostatní části, tedy zbytek obrazu. V této kapitole si nyní ukážeme rozdělení obrazu na dvouúrovňový, černobílý snímek pomocí prahování.

Prahování se používá k převedení obrazu s více úrovněmi jasu, jako je například šedotónový obraz, na obraz, ve kterém se vyskytují pouze dvě úrovně jasu a to černá a bílá. Bodům s hodnotou jasu větší než je určená hodnota, tedy práh, je přiřazena barva s hodnotou 1, případně 255, ostatním bodům je pak přiřazené hodnota 0. Metoda prahování může být buď globální nebo lokální.

Práh je možno stanovit různými způsoby. Na základě výběru hodnot jasu v obraze nebo z histogramu je možno vybrat práh tak, že zkoušením několika různých hodnot prahů najdeme hodnotu, která dává nejlepší výsledek. Problém nastává v případě hromadného zpracování tištěného textu. V takovém případě je tento postup naprosto nepoužitelný, protože vyžaduje zásah uživatele. Existují samozřejmě také metody pro nalezení automatické prahové hodnoty.

Globální prahování

Český název byl převzat z anglického Global Thresholding. Vstupem operace prahování je typicky obraz v odstínech šedi. Metoda je charakteristická tím, že do ní vstupuje jediný parametr známý jako globální prahová hodnota. V tomto případě je každý pixel obrazu porovnáván s touto prahovou hodnotou. Pokud je intenzita bodu vyšší než prahová hodnota, je barva bodu nastavena na bílou, v opačném případě je bodu přiřazena černá barva.

Ovšem ne všechny obrázky mohou být takto segmentovány na objekty a pozadí. Zda může být obraz správně segmentován tímto způsobem nebo ne, může být zjištěno vyšetřením jeho histogramu intenzity jasu.

Jednou z hlavních funkcí histogramu je určení prahové hodnoty, kterou je vhodné použít pro převádění obrazu v odstínech šedi do dvoubarevné podoby pomocí prahování. Pokud je obrázek vhodný pro globální prahování, bude histogram bi-modální. Bi-modální histogram znamená to, že intenzity jednotlivých bodů budou seskupeny okolo dvou dobře separovaných hodnot. Vhodná prahová hodnota pro oddělení těchto dvou skupin je někde mezi těmi dvěma vrcholy v histogramu. Pokud

ovšem rozdělení intenzit nemá tuto charakteristiku, pak je nepravděpodobné, že globální prahování bude dávat pro zpracovávaný obraz přijatelný výsledek.

Pokud bychom chtěli převést obrázek pomocí metody globálního prahování, která pro jeho zpracování není vhodná, bude výsledný obraz nepoužitelný.

Obrázek 22.:Obrázek před a po použití globálního prahování

Adaptivní prahování

Pokud se v obraze vyskytují různé stíny a šum, je potřeba zamyslet se nad určováním prahové hodnoty. Prahovou hodnotu není možné určit globálně, protože jedna hodnota není vhodná pro celý obraz jako celek. Z toho vyplývá, že pro jeden obraz musí být určeno více prahových hodnot. Obecně je práh počítán pro každý pixel originálního obrazu, tato hodnota je poté použita pro porovnávání s hodnotou daného pixelu a vzniká nový obraz.

$$T = T[x, y, p(x, y), f(x, y)]$$

Vzorec 24.: Definice prahové hodnoty

Kde $f(x, y)$ je stupeň šedi bodu (x, y) originálního obrazu a $p(x, y)$ je určitá lokální vlastnost tohoto bodu. Pokud hodnota T závisí jen na odstínu šedi v daném bodě, potom z něj vzniká obyčejná globální prahová hodnota. Zvláštní pozornost by měla být věnována faktoru $p(x, y)$, který byl popsán jako vlastnost určitého bodu. Ve skutečnosti je tato vlastnost jednou z nejdůležitějších prvků při počítání prahové hodnoty pro bod v obraze. Výpočet této vlastnosti je obvykle založen na okolí bodu, aby bylo možné vzít v úvahu vliv osvětlení a šumu. Vlastnost může být například průměrná hodnota v předdefinovaném okolí, jehož je daný bod středem.

Metoda adaptivního prahování je tedy typická tím, že mění velikost prahové hodnoty dynamicky nad obrázkem, který zpracovává. Tato lépe propracovaná varianta prahování umožňuje přizpůsobit se stavu osvětlení v obraze, tj. velkým změnám intenzity osvětlení nebo stínům.

Pro každý pixel v obraze je počítána nová prahová hodnota. Převod poté probíhá stejným způsobem jako u prahování s globální hodnotou prahu: pokud je hodnota pixelu nižší než prahová, je jeho barva nastavena na barvu popředí, v opačném případě je bodu přiřazena hodnota pozadí.

Pro hledání prahové hodnoty se používá několik postupů. Mezi hlavní patří algoritmus Otsu a lokální prahování. Tyto metody využívají společného předpokladu a sice, že u menších oblastí obrazu je pravděpodobnější, že budou mít uniformní osvětlení. Proto se menší oblasti jeví jako vhodnější pro určování prahové hodnoty.

Algoritmus Otsu

Nejčastěji používanou metodou pro nalezení optimální prahové hodnoty je metoda, kterou navrhl Nobyuki Otsu. Histogram budeme považovat za funkci hustoty pravděpodobnosti. Předpokládáme, že histogram lze aproximovat dvěma křivkami p_1 a p_2, kde křivky představují dvě rozdělení reprezentující popředí a pozadí. Základní myšlenkou je nalézt tzv. „správný střed“, tedy práh, tak aby rozdělení byla co nejdál od sebe. V některých případech může nastat chyba při nesprávném určení popředí jako pozadí nebo pozadí jako popředí.

Otsuova metoda hledá hodnotu prahu T na základě informace z histogramu. V závislosti na hodnotě prahu vzniknou dvě množiny bodů. Množina C_0 s úrovněmi $[0, 1, \dots, k-1]$ odpovídající pozadí a množina C_1 s úrovněmi $[k, k+1, \dots L-1]$ odpovídající popředí. Množinu pro pozadí charakterizují sumy hodnot ω_0, μ_0 a množinu pro popředí charakterizují sumy hodnot ω_1, μ_1, které závisí na hodnotě prahu T. [4]

$$\omega_0 = \sum_{k=0}^{T-1} p(k)$$

$$\mu_0 = \sum_{k=0}^{T-1} kp(k) \,/\, \omega_0$$

$$\omega_1 = \sum_{k=T}^{L-1} p(k)$$

$$\mu_1 = \sum_{k=T}^{L-1} kp(k) \,/\, \omega_1$$

Vzorec 21.:Sumy hodnot pro množiny

Dále se musí spočítat ještě střední hodnotu v původním rozdělení. Výslednou hodnotu dále použijeme pro nalezení optimálního prahu.

$$\mu_T = \sum_{k=0}^{L-1} kp(k)$$

Vzorec 22.: Střední hodnota pro původní rozdělení

Nyní můžeme hledat optimální prah jako takové $k = T$, které maximalizuje výraz:

$$\sigma^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2$$

Vzorec 23.: Výpočet mezitřídního rozptylu pro stanovení optimálního prahu

Metoda Otsu prahování vykazuje opravdu kvalitní výsledky a proto je vhodná pro použití při hromadném zpracovávání dokumentů. Díky vysoké rychlosti samotného algoritmu a nalezení optimálního prahu je tato metoda jednou z nejpoužívanějších.

Obrázek 23.: Obrázek před a po použití Otsu metody

Lokální prahování

Metoda lokálního prahování hledá prahovou hodnotu pro každý pixel statistickým vyšetřením intenzit jasu v lokálním okolí bodu. Druh statistiky, který je nejvhodnější, je do značné míry závislý na vstupním obrazu. Jednoduché a rychlé funkce pro tuto operaci počítají s hodnotou T, kterou je možno chápat třemi způsoby. Prvním způsobem je určení střední hodnoty z intenzit jasu v lokálním okolí bodu ($T = střední\ hodnota$). Dalším je určení hodnoty zvané medián, což je prostřední hodnota z intenzit jasu v okolí ($T = medián$). Posledním a nejjednodušším způsobem je vypočtení hodnoty T jako $T = (\max - min)\ /\ \mathbf{2}$, kde max je rovno maximální hodnotě jasu v okolí bodu a min je rovno minimální hodnotě jasu v daném okolí.

Problémem je velikost okolí, která musí být tak rozlehlá, aby pokryla dostatečně mnoho pixelů, která jsou součástí objektů (písmen, obrázků) i pixelů pozadí. V opačném případě by byla zvolena špatná prahová hodnota a výsledek by byl nepoužitelný. Pokud ale vybereme příliš velké okolí bodu, můžeme porušit předpoklad, že osvětlení v lokálním okolí bodu se příliš nemění.

Pokud provedeme porovnání s globálním prahováním, u kterého se určuje pouze jediná prahová hodnota, aplikovaná na celý obraz a které je vlastně použitelné pouze pokud histogram intenzity osvětlení v obraze splňuje určité požadavky. Oproti tomu lokální prahování volí pro každý bod jinou hodnotu prahu a tak může být použito pro obrazy, jejichž histogram intenzity osvětlení neobsahuje dva oddělené vrcholy.

Metoda lokálního prahování je určena pro zpracování obrazů obsahujících text. Pokud však obraz obsahuje velký objekt, dochází k chybnému převodu. Barva všech pixelů uvnitř objektu, které nemají ve svém okolí žádný pixel náležící do pozadí, je chybně nastavena na barvu pozadí. Takové chování lze ovlivnit změnou velikosti okolí bodu na vysokou hodnotu. Ovšem výpočty pro okolí velké například 140x140 pixelů mohou být časově náročné. Z tohoto důvodu se metoda lokálního prahování používá především u obrazů, u nichž je nutné získat co nejkvalitnější prahovaný obraz. U problematiky OCR je sice kladen důraz na kvalitní prahovaný obraz, ale ve většině případů lez tento požadavek vypustit. Vezmeme-li v potaz časovou náročnost pro jeden obraz, tak nemůžeme počítat s přijatelnou časovou náročností například pro 50 obrazů. Pro hromadné zpracování textu se dají mnohdy použít časově méně náročné metody.

P - procentní prahování

V některých případech se může poměrně jednoduše odhadnout, jakou plochu (p %) zaujímá popředí vzhledem k pozadí. Díky tomu můžeme získat během krátké časové náročnosti kvalitní výsledky. Vezmeme-li v potaz, že hodnota p vyjadřuje procentuální zastoupení popředí objektu, vzhledem k celkové ploše obrazu, můžeme pro výpočet p – procentního prahování použít následující vzorec:

$$\frac{p}{100} = \sum_{k=0}^{T} \frac{n_k}{N}$$

Vzorec 24.: Vzorec pro nalezení prahové hodnoty u p – procentního prahování

Nechť p je hodnota kolik procent zabírá popředí, $k \in \langle 0, 1, \dots, L-1 \rangle$ je k-tá úroveň jasu, n_k je počet výskytů dané úrovně jasu j v prohledávané oblasti a N je počet pixelů v prohledávané oblasti (masky). Pro každé posunutí masky hledáme takovou hodnotu T, aby odpovídala výše uvedenému vzorci.

Obrázek 24.: Obrázek před a po použití metody p – procentního prahování

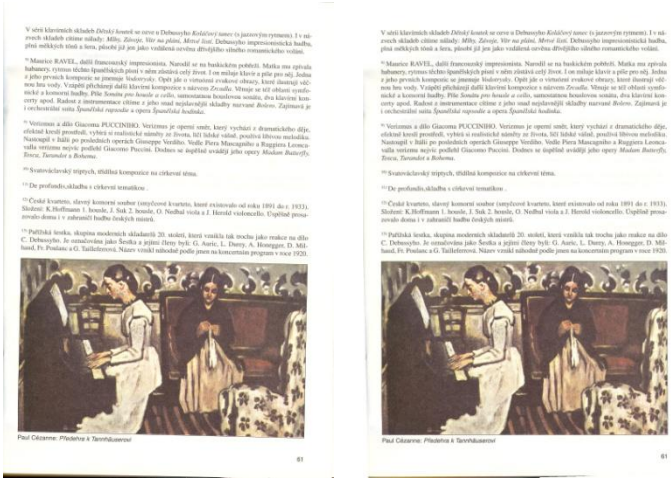
2.2.7 Deformovaný vstupní obraz

Pokud jako vstupní materiál pro zpracování tištěného obrazu použijeme oskenovaný obraz, může se ve většině případů stát, že je výsledný obraz v počítači zakřivený. To je způsobeno právě špatným oskenováním. U lepšího softwaru pro samotné oskenovaní jsou k dispozici nástroje, odstraňující tento nedostatek. Takže i když jste umístili obrázek pro oskenování do skeneru nezarovnaně, můžete tento problém vyřešit.

Nepříjemnosti ovšem nastanou v případě, že jste chybně oskenovaný soubor získali nebo nemáte k dispozici potřebné nástroje pro provedení patřičných úprav. V takovém případě se musí tzv. „zarovnání“ celého obrazu provádět před veškerými úpravami vad a nedokonalostí vstupního materiálu. To je způsobeno hlavně nutností narovnávat přímo zdrojový obraz a ne až jeho upravenou formu. Například prahovaný zakřivený obraz už neobsahuje takové barevné informace, se kterými pracuje narovnací algoritmus. Naštěstí jsou k dispozici algoritmy, které řeší zarovnání obrazu podle nahnutí textu v obraze, což je přesně potřebná metoda pro úpravu takového problému. U takových algoritmů se většinou počítá s použitím křivě oskenovaných obrazů. Jednoduše lze tedy zjistit, pod jakým úhlem je potřeba provádět narovnání a tím pádem upravení obrazu před prováděním například prahování.

Detekce dolní hrany

Metoda detekce dolní hrany je nejjednodušším zarovnávacím algoritmem před samotnou fází OCR. U této metody se počítá s tím, že je vstupní obraz špatně oskenovaný a že jde toto chybné oskenovaná zjistit podle úhlu naklonění dolního okraje. Právě pokud je oskenovaný obraz nahnutý, vytvoří se pod dolním okrajem tmavý pruh, který svírá s dolní hranou obrázku určitý úhel. Pomocí algoritmu se spočítá chybné prohnutí obrazu a následně se provede potřebné narovnání. Takto narovnaný obraz je pak zarovnaný právě podle dolního okraje obrázku. Nevýhodou pro tento algoritmus je v některých případech záměna dolní hrany obrazu s obrázkem v obraze. To může zapříčinit nevhodné narovnání a tím pádem naprostou nepoužitelnost obrazu pro zpracování. Ve většině případů se používá klasické ruční zarovnání uživatelem.



Obrázek 25.: Příklad křivého vstupního obrazu a výsledku metody detekce dolní hrany

Prohnutý text

Problém prohnutého textu nastává většinou u oskenovaných knih. Kde je obraz u strany, která byla u vazby, prohnutý. Také tento problém lze vyřešit. Dokonce podobně jako problém zakřiveného obrazu. Zde by se ovšem musel analyzovat postupně každý řádek textu a na výsledek by se čekalo poměrně dlouhou dobu. Nejlepším a časově nenáročným řešením se jeví použití tzv. „rozbalovacího“ algoritmu, který potřebuje pro svůj vstup znát mimo zdrojového obrazce s prohnutým textem také rádius a střed, pod kterým se text na obraze prohýbá. Následně se provede narovnání prohnutých řádků. Problém prohnutého textu je ale nejčastějším problémem, proč zvažovat skenování za použití lepšího skeneru. V dnešní době existuje řešení skenování knih, které právě eliminuje prohnutí textu na straně vazby. Právě tato metoda dokáže eliminovat nevhodné prohnutí a tím pádem zaručuje možnost rychlého digitálního zpracování celé knihy.

Text na kružnici

Současným nejrozšířenějším problémem u problematiky digitálního zpracování textu je problém textu na kružnici. Problém, kdy text není na rovném řádku a kdy by zpracování po jednom různě natočeném znaku zabral spoustu času.

Nejlepším řešením je takový text tzv. „narovnat“ do řádku a tím pádem usnadnit práci samotnému algoritmu pro rozpoznání znaků. V případě narovnání textu do řádku se totiž s takovým textem pracuje jako s normálním textem z dokumentu. Problém zakřiveného textu se v současné době řeší také na různých konferencích a v časopisech, zaměřených na OCR. Mezi takové příklady patří například poštovní razítko nebo text na minci. V dnešní době neexistuje kvalitní komerční nástroj, který by vedle zpracování tištěného textu řešil problematiku tzv. „narovnáni textu na kružnici“. I když je k dispozici několik kvalitních algoritmů, které tento problém řeší.

Nejlepším a časově nenáročným řešením je nalezení středu kružnice, na které je text napsán a poté určení poloměru dané kružnice. V další fázi se některé postupy liší. U některých se řešení doplňuje o krok získání pouze části zakřiveného textu, u jiných se tento krok vynechává a postupuje se k narovnání celého obrazu. Následně se provede narovnání výběru podle úhlů a podle zadaného poloměru kružnice. Výsledkem je pak narovnaný pruh obrazu, obsahující narovnaný a tudíž dále zpracovatelný text. Takový obraz pak poslouží jako vstupní obraz pro fázi OCR. Nevýhodou řešení je nutná interakce uživatele s aplikací pro co nejdetailnější určení středu a poloměru kružnice. [7], [8]

Obrázek 26.: Příklad narovnaného textu z kružnice

3. Běžně používané techniky a algoritmy pro OCR

Základním principem u problematiky OCR je v první řadě naučit systém základním případům. Jak tyto případy vypadají a jak mohou nastat. U OCR je za základní typ považován znak. Takže písmena, číslice a některé speciální symboly jako jsou čárka, vykřičník, otazník a mnoho dalších jsou myšleny základními typy.

Nejrozšířenější metodou u softwarů řešících zpracování tištěných dokumentů je tzv. „učení systému“. Učení systému se provádí dodáním příkladů znaků ve všech rozdílných typech. Podle těchto příkladů si systém vyrobí prototypy nebo popis každé třídy každého znaku. Pří rozpoznávání se každý neznámý znak porovnává s dříve opatřeným popisem a je stanovena třída, která koresponduje s tímto znakem.

Různé OCR algoritmy byly navrženy k dosažení lepších výsledků rozpoznávání. Tyto algoritmy pracují například se shodou šablon, popisy obrazců, geometrickými vlastnostmi, a obrazy založenými na invariantách.

Ze všech OCR algoritmů jsou právě ty, které jsou založeny na invariantách, zvlášť zajímavé. Takové algoritmy mohou totiž výrazně zrychlit výsledné rozpoznání. To je způsobeno pamatováním si již rozpoznaných objektů, ať už různě posunutých, pootočených nebo zmenšených, a v případě velmi podobného zdroje je nalezení výsledku s velkou pravděpodobností správné. Rozpoznání obrazu založené na této metodě zahrnuje tři důležité kroky:

- Rozpoznání objektu
- Rozsah podobnosti objektu
- Indexování objektu

Z těchto tří kroků je právě rozpoznání objektu tím nejdůležitějším. Existuje mnoho metod, které provádějí rozpoznávání a popisování objektu, ale všechny se dělí do dvou kategorií. První pracuje s hranicemi a druhá s oblastmi invariant. Algoritmus, pracující s hranicemi invariant, prozkoumává informace obsažené v obrysu zkoumaného tvaru. Obvyklé hranice invariant obsahují kód řetězce a Fourierovy deskriptory. V případě oblastí invariant jsou brány v potaz veškeré pixely v objektu, pro získání matematické reprezentace.

Ovlivnit kvalitu výstupu OCR algoritmu jde především co nejkvalitnějším vstupním obrazem. Pokud by byl vstupní obraz rozmazaný, nekvalitně oskenovaný nebo různě znehodnocený, rozhodně by to výsledný převod negativně ovlivnilo. Z tohoto důvodu je pro převod volit co nejkvalitnější obraz. Ovšem ne vždy lze opravdu kvalitní obraz získat. V takovém případě přicházejí na řadu různé algoritmy, které řeší úpravu nekvalitního vstupního materiálu. Tento problém se řeší v předchozí kapitole. Hlavní cíl, který si klade každý OCR software, je dosažení co nejpřesnějšího převodu. Kvalitní převod se zde hodnotí počtem chyb na řádek. Někdy ani takové hodnocení neznačí kvalitní nástroj a tak se zavádí ještě náročnější hodnocení, kde se chyby počítají v poměru počtu chyb na

stránku. Označit software za kvalitní znamená přesnost rozpoznání 99%, a tedy nalezení maximálně 5 chyb na stránku. Z tohoto hlediska musí software umět pracovat jak s kvalitním obrazem, tak s nekvalitním obrazem, který je později upraven řadou algoritmů do použitelné podoby.

Základním prvkem všech těchto návrhů je definice sady matematických funkcí pro reprezentaci obrazu a redukci dat. Obvykle je zapotřebí provádět další transformace k dosažení požadovaných vlastností invariantního obrazu. [5]

3.1 Obrazové invarianty

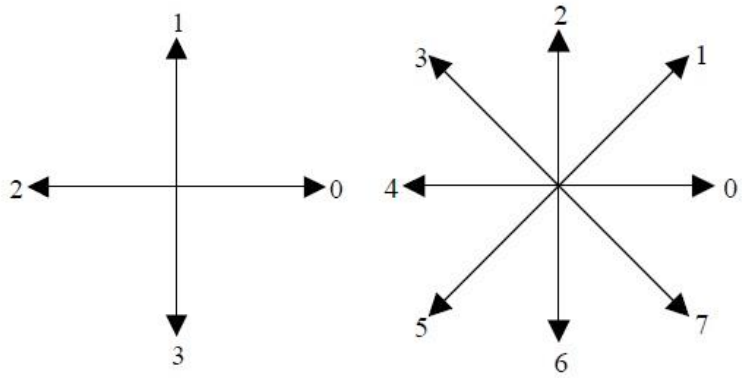
Hlavním problémem u OCR je automatické rozpoznání znaku v obraze bez ohledu na jeho pozici, velikost a orientaci. Aby bylo možné rozpoznat různé varianty stejného znaku, je nutné použít obrazové funkce, které jsou invariantní k některým transformacím. Obrazové invarianty jsou vlastnosti, které mají přibližně stejné hodnoty pro vzorky stejného obrazu, které jsou například různě velikostně rozdílné, otočené, zkosené nebo rozmazané.

3.1.1 Invarianty založené na hranicích

Hranice je jednou z nejjednodušších a důležitých vlastností obrazu. Lidé mají sklon vnímat scény složené z jednotlivých objektů, u kterých dokážou rozeznat nejlépe právě jejich hranice. Pokud pomineme implementaci nalezení hranice objektu, je také poměrně jednoduché takové hranice spočítat.

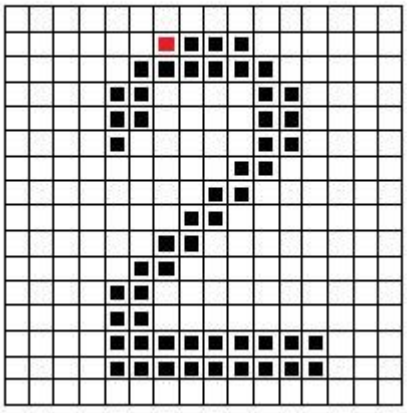
Kód řetězce

Kód řetězce byl zaveden Freemanem jako prostředek reprezentace hran nebo hranic tvarů, spojených úseky přímek různých délek a směrů. Tento kód se skládá ze dvou částí. První částí jsou výchozí souřadnice a tou druhou je řetězec kódů, vyjadřujících relevantní pozici výchozího pixelu a následných pixelů. Je generován pomocí změny směru spojených pixelů obsažených v hranici. Výsledný kód je založen na 4 nebo 8 směrech spojených úseků. Směr každého úseku může být určen například pomocí následujícího schématu:



Obrázek 27.: Určení 4 nebo 8 směrů pro spojení úseků kódu řetězce

Kód řetězce závisí u uzavřených hranic právě na výchozím bodu hranice. Je ovšem možné stanovit tuto závislost jako invariantní díky normalizaci kódu řetězce: Kód řetězce může být považován za kruhovou posloupnost čísel. Díky tomu se nový výchozí bod stanoví jako nejmenší hodnota z této kruhové posloupnosti. Například podle znaku na [Obrázek 28.] můžete určit kód řetězce podle osmisměrného schématu z [Obrázek 27.].



Obrázek 28.: Znak v mřížce pro možnost určení kódu řetězce

Výchozí bod je na obrázku rozlišen. Nyní se při určování kódu řetězce postupuje podle schématu [Obrázek 27.].

00007766555555660000000644444444222111111223444565221 1

Vzorec 25.: Výsledný kód řetězce

Kód řetězce může být samozřejmě normalizován také vůči rotaci, použitím tzv. „první diference“. První diference je určena spočítáním změny směru dvou sousedních pixelů. Výchozí bod může být

spočítán jako součet změny směru mezi koncovým a počátečním pixelem kódu řetězce. Hodnota první diference bude udržovat invariantní kód řetězce vůči rotaci, pouze pokud bude hranice samotná invariantní vůči rotaci.

Fourierovy deskriptory

Fourierovy deskriptory byly popsány Zahnem a Roskiesem, Persoonem a Fuem jako prostředek k popsání tvaru uzavřeného rovinného obrazce. Hranice s může být nalezena a převzorkována proti směru hodinových ručiček k získání rovnoměrně rozloženého počtu K bodů. Každý bod může mít vyjádřeny souřadnice jako $(x_0, y_0), (x_1, y_1), \dots, (x_{K-1}, y_{K-1})$. Jednotlivé souřadnice mohou být zobecněny na zápis $x(k) = x_k$ a $y(k) = y_k$. Za těchto podmínek může být hranice vyjádřena jako sekvence komplexních čísel. [5]

$$s(k) = x(k) + j\, y(k), \quad k = 0, 1, 2, \dots, K-1$$

Vzorec 26.: Sekvence komplexních čísel pro Fourierovy deskriptory

To znamená, že x-ová osa je považována za reálnou osu a y-ová osa je považována za imaginární osu se sekvencí komplexních čísel. Koeficienty Diskrétní Fourierovy Transformace (DFT) této komplexní sekvence jsou označeny $z(u)$:

$$z(u) = \frac{1}{K} \sum_{k=0}^{K-1} s(k) e^{-j\,2\pi u k\,/K}, \quad u = 0, 1, 2, \dots, K-1$$

Vzorec 27.: Koeficient DFT

Komplexní koeficienty $z(u)$ se nazývají Fourierovy deskriptory hranic. Fourierovy deskriptory mají výhodu v tom, že snižují 2-D problém na 1-D problém. Významné Fourierovy deskriptory mohou být použity jako základ pro klasifikaci různých hranic tvarů.

Jak bylo uvedeno výše, chceme dosáhnout toho, aby obrazové invarianty byly insenzitivní vůči změně měřítka, posunutí a rotaci. Fourierovy deskriptory nejsou přímo insenzitivní vůči těmto obrazovým transformacím, ale podle následujících operací se mohou insenzitivitě co nejvíce přiblížit.

Transforamce	**Hranice**	**Fourierův deskriptor**
Totožnost	$s(k)$	$z(u)$
Rotace	$s_r(k) = s(k)e^{j\theta}$	$z_r(u) = z(u)e^{j\theta}$
Posunutí	$s_t(k) = s(k)\Delta_{xy}$	$z_t(u) = z(u) + \Delta_{xy}\,\delta(u)$
Změna měřítka	$s_s(k) = \alpha s(k)$	$z_s(u) = \alpha z(u)$
Výchozí bod	$s_p(k) = s(k - k_0)$	$z_p(u) = z(u)e^{-j\,2\pi u\, k_0/K}$

Tabulka 1.: Základní vlastnosti Fourierových deskriptorů podle některých transformací

Symbol Δ_{xy} lze vyjádřit jako $\Delta_{xy} = \Delta x + j\Delta y$ a $\delta(u)$ je impulzní funkce, která má na počátku nenulovou hodnotu a poté už má stále hodnotu 0.

Z tabulky [Tabulka 1.] můžeme vyčíst, že veličina $z(u)$ je invariantní vůči rotaci obrazu, což lze vyčíst z následujícího vzorce:

$$|z_r(u)| = \left|z(u)e^{j\theta}\right| = |z(u)| \cdot \left|e^{j\theta}\right| = |z(u)| \cdot 1 = |z(u)|$$

Vzorec 28.: Vyjádření invariance vůči rotaci obrazu

Posunutí obrazu závisí na přidání odpovídajícího posunutí každého vzorku souřadnic hranice. Posunutí nemá žádný vliv na deskriptory s výjimkou hodnoty $u = 0$, právě kvůli impulzní funkci $\delta(u)$. První složka Fourierových deskriptorů závisí pouze na pozici tvaru, což není vůbec použitelné pro popis tvaru a proto může být tato složka vypuštěna.

Invariance vůči změně měřítka může být dosaženo poměrem dvou koeficientů, takže se můžeme zbavit parametru α.

Veličina $z(u)$ je invariantní vůči změně výchozího bodu právě díky následujícímu vztahu:

$$\left|z_p(u)\right| = \left|\, z(u)e^{-j\,2\pi u k_0}\right| = |z(u)| \cdot \left|\, e^{-j\,2\pi u k_0}\right| = |z(u)| \cdot 1 = |z(u)|$$

Vzorec 29.: Vyjádření invariance vůči změně výchozího bodu

Pokud vezmeme v potaz všechny výše uvedené analýzy, můžeme použít vzorec:

$$c(u-2) = \frac{|z(u)|}{|z(1)|}, \quad u = 2, 3, \ldots, K-1$$

Vzorec 30.: Výsledná invariance Fourierových deskriptorů

pro získání invariance Fourierových deskriptorů vůči současné rotaci, posunutí a změně měřítka.

3.1.2 Invarianty založené na oblastech

Invarianty založené na hranicích, jako jsou kód řetězce nebo Fourierovy deskriptory, zkoumají pouze obrysovou informaci. Nemohou zachytit vnitřní obsah tvaru. Na druhé straně si tyto metody neporadí s disjunktními tvary, kdy nemusí být k dispozici jediná uzavřená hranice. Proto je jejich použití omezené. U invariant založených na oblastech jsou brány v úvahu všechny pixely obrazu pro reprezentaci tvaru. Protože takové invarianty kombinují informace z celé oblasti obrazu, než využívání informací pouze z hraničních pixelů, mohou zachytit více informací z obrazu. Invarianty založené na oblastech mohou být také použity k popsání disjunktních tvarů.

Invarianty založené na momentech jsou nejběžněji používanými invariantami, které jsou založeny na oblastech, a které se používají jako vzory pro funkce v mnoha aplikacích. V roce 1961 představil Hu jako první sadu invariant založených na momentech používajících nelineární kombinace pravidelných momentů. Huovy invarianty mají potřebné vlastnosti invariance pro posunutí, změnu měřítka a rotaci

obrazu. Bylo ovšem zjištěno, že výpočet vyšších stupňů Huových invariantních momentů, je poměrně složitý a také obtížný pro rekonstrukci obrazu z Huových invariant. Pro vyřešení těchto problémů byl navržen pojem ortogonální momenty pro obnovení obrazu z invariantních momentů založených na teorii ortogonálních polynomů. Byly představeny Zernikovy momenty, které umožňují jednoduché sestavení nezávislých invariantních momentů libovolného stupně. Pseudo-Zernikovy momenty jsou další formou ortogonálních momentů, navržené Tehem a Chinem, které nejsou tak citlivé na šum jako běžné Zernikovy momenty.

Invarianty založené na momentech prozkoumávají informace v rámci celého obrazu a nikoliv poskytování informací pouze z bodů hranice. Mohou zachytit některé globální vlastnosti, které nelze zachytit z hraniční reprezentace, jako je například celková orientace obrazu. [5]

Normální a centrální momenty

Normální momenty, které jsou také někdy označovány jako geometrické momenty, jsou definovány takto:

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)dxdy, \quad p,q = 0,1,2,\dots$$

Vzorec 31.: Definice normálních momentů

kde m_{pq} je $(p+q)$. stupeň momentu spojité obrazové funkce $f(x,y)$.

Centrální momenty jsou definovány jako:

$$\mu_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q f(x,y)dxdy, \quad p,q = 0,1,2,\dots$$

Vzorec 32.: Definice centrálních momentů

kde $\bar{x} = m_{10} / m_{00}$ a $\bar{y} = m_{01} / m_{00}$, jsou těžištěm obrazu.

U digitálního obrazu se integrály nahrazují sumou, takže m_{pq} je upraveno [6]:

$$m_{pq} = \sum_{x}\sum_{y} x^p y^q f(x,y)dxdy, \quad p,q = 0,1,2,\dots$$

Vzorec 33.: Definice normálních momentů pro digitální obraz

Samozřejmě se stejně změní také definice pro centrální momenty digitálního obrazu. Centrální momenty jsou počítány pomocí těžiště obrazu a jsou ekvivalentní k normálním momentům s posunutým středem obrazu právě do těžiště obrazu. Proto jsou centrální momenty invariantní vůči posunutí.

U afinních transformací byla změna měřítka definována jako:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Vzorec 34.: Definice změny měřítka u afinních transformací

Abychom získali invarianci vůči změně měřítka, dosadíme funkci $f'(x', y')$, která reprezentuje obrazovou funkci $f = (x, y)$ po změně měřítka obrazu $S_x = S_y = \alpha$. Takže $f'(x', y') = f'(\alpha x, \alpha y) = f(x, y)$ a $x' = \alpha x$, $y' = \alpha y$. Po dosazení těchto úprav do původní definice získáme vztah:

$$\begin{aligned} m'_{pq} &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x'^p y'^q f'(x', y') dx' dy' \\ &= \alpha^{p+q+2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx dy \\ &= \alpha^{p+q+2} m_{pq} \end{aligned}$$

Vzorec 35.: Definice invariance vůči změně měřítka normálních momentů

Podobně lze tuto úpravu aplikovat také na definici centrálních momentů.

$$\mu'_{pq} = \alpha^{p+q+2} \mu_{pq}, \quad \mu'_{00} = \alpha^2 \mu_{00}$$

Vzorec 36.: Definice invariance vůči změně měřítka centrálních momentů

Můžeme také definovat normalizovaný centrální moment vztahem:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \quad \gamma = p + q + 2/2 \, , p + q = 2, 3, \ldots$$

Vzorec 37.: Definice normalizovaného centrálního momentu

Normalizovaný centrální moment η_{pq} je invariantní vůči změně měřítka, což je vyjádřeno v následujícím vztahu:

$$\eta'_{pq} = \frac{\mu'_{pq}}{\mu'^{\gamma}_{00}} = \frac{\alpha^{p+q+2} \mu_{pq}}{\alpha^{2\gamma} \mu_{00}^{\gamma}} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} = \eta_{pq}$$

Vzorec 38.: Vyjádření invariance vůči změně měřítka u normalizovaného centrálního momentu

Pokud shrneme veškeré výše uvedené definice, pak určíme, že normalizované centrální momenty jsou invariantní vůči posunutí a změně měřítka.

Sedm Huových invariantních momentů

Tyto momenty jsou založeny na normalizovaných centrálních momentech. Hu představil sedm nelineárních funkcí, která jsou invariantní vůči posunutí, změně měřítka a rotaci. Těchto sedm momentů je definováno následovně:

$$\phi_1 = \eta_{20} + \eta_{02}$$
$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$
$$\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$
$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{30})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

Vzorec 39.: Definice sedmi Huových invariantních momentů

Sedm Huových invariantních momentů je invariantních vůči transformacím obrazu, včetně posunutí, změně měřítka a rotaci. Nicméně sada těchto invariantních momentů není invariantních vůči změnám kontrastu.

Sedm Huových invariantních momentů bylo široce využíváno v rozpoznávání a jejich výkonnost byla hodnocena na základě různých situací deformace, včetně rozmazání, prostorového zhoršování, náhodného šumu, zkosení a transformace pohledu.

Protože tyto invariantní momenty berou každý pixel obrazu pro výpočet, je také výpočetní čas mnohem vyšší než u výpočtu invariant založených na hranicích. [5]

Zernikovy momenty

Zernike představil sadu komplexních polynomů $\{V_{nm}(x, y)\}$, které tvoří úplnou ortogonální sadu z jednotky kruhu $x^2 + y^2 \leq 1$ v polárních souřadnicích. Polynomy jsou definovány následovně:

$$V_{nm}(x, y) = V_{nm}(\rho, \theta) = R_{nm}(\rho)e^{im\theta}$$

Vzorec 40.: Definice polynomů Zernikových momentů

kde n nabývá celočíselných kladných hodnot nebo 0; m nabývá celočíselných hodnot, ale navíc je omezeno vztahy $n - |m|$ a $|m| \leq n$; ρ je velikost vektoru od počátku po pixel (x, y); θ je úhel mezi vektorem ρ a osou x v proti směru hodinových ručiček; $R_{nm}(\rho)$ je radiální polynom definovaný následovně:

$$R_{nm}(\rho) = \sum_{s=0}^{(n-|m|)/2} (-1)^s \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!}\rho^{n-2s}$$

Vzorec 41.: Definice radiálního polynomu

Zernikův moment n-tého stupně s m opakováním pro funkci $f(x,y)$, je definován:

$$A_{nm} = \frac{n+1}{\pi}\iint_{x^2+y^2\leq 1} f(x,y)V_{nm}^*(x,y)dxdy, \quad V_{nm}^*(x,y) = V_{n,-m}(x,y)$$

Vzorec 42.: Definice Zernikova momentu n-tého stupně s m opakováním

Pro výpočet Zernikova momentu pro digitální obraz musíme pouze zaměnit dvojitý integrál za sumy. Takže upravená definice pro digitální obraz je následující:

$$A_{nm} = \frac{n+1}{\pi}\sum_x\sum_y f(x,y)V_{nm}^*(x,y), \quad x^2+y^2 \leq 1$$

Vzorec 43.: Definice Zernikova momentu n-tého stupně s m opakováním pro digitální obraz

Definované funkce Zernikových momentů jsou invariantní pouze vůči rotaci. Pro dosažení invariance vůči změně měřítka a posunu musí být obraz nejprve normalizován, použitím normálních Zernikových momentů.

Invariance vůči posunutí je dosaženo převodem původního obrazu $f(x,y)$ na $f(x+\bar{x}, y+\bar{y})$, kde $\bar{x} = \frac{m_{10}}{m_{00}}$ a $\bar{y} = \frac{m_{01}}{m_{00}}$. Jinak řečeno, musí být střed původního obrazu přesunut do těžiště obrazu před výpočtem Zernikova momentu.

Invariance vůči změně měřítka je dosaženo rozšířením nebo zúžením každého tvaru, takže první normální moment obrazu m'_{00} odpovídá předdefinované hodnotě β. Pro binární obraz odpovídá m_{00} celkovému počtu pixelů tvarů v obraze. A z rovnice [Vzorec 35.] pro obraz se změněným měřítkem $f(\alpha x, \alpha y)$ je normální moment $m'_{pq} = \alpha^{p+q+2}m_{pq}$, m_{pq} je tedy normální moment pro $f(x,y)$. A protože cílem je určit $m'_{00} = \beta$, můžeme ponechat $\alpha = \sqrt{\beta / m_{00}}$. Substitucí $\alpha = \sqrt{\beta / m_{00}}$ v m'_{00} určíme $m'_{00} = \alpha^2 m_{00} = \beta$.

Předpokládáme, že známe všechny Zernikovy momenty A_{nm} z $f(x,y)$ až do N-tého stupně. Vzhledem k ortogonálním vlastnostem Zernikových momentů, můžeme obnovit obraz založený na sadě Zernikových momentů podle vztahu:

$$f'(x,y) = \sum_{n=0}^{N}\sum_m A_{nm}V_{nm}(x,y)$$

Vzorec 43.: Vztah pro obnovu obrazu pomocí Zernikových momentů

Pseudo-Zernikovy momenty

Zernikovy momenty, které jsou invariantní vůči rotaci, jsou polynomiální v x a y. Podobná ortogonální sada polynomů v x, y a ρ, odvozená Bhation a Wolfem, má podobné vlastnosti jako Zernikovy momenty. Tato sada polynomů, která se jmenuje Pseudo-Zernikovy momenty od Teha a China, se liší od Zernikových momentů v reálných radiálních polynomech, definovaných následovně:

$$R'_{nm}(\rho) = \sum_{s=0}^{n-|m|} (-1)^s \frac{(2n+1-s)!}{s!\,(n-|m|-s)!\,(n+|m|+1-s)!}\rho^{n-s}$$

Vzorec 44.: Definice radiálního polynomu pro Pseudo-Zernikovy momenty

Pseudo-Zernikovy momenty jsou tedy definovány:

$$A'_{nm} = \frac{n+1}{\pi}\sum_x\sum_y f(x,y)R'_{nm}(x,y)e^{-im\theta}$$

Vzorec 45.: Definice Pseudo-Zernikových momentů

kde n nabývá celočíselných kladných hodnot nebo 0; m nabývá celočíselných hodnot, ale navíc je omezeno vztahem $|m| \leq n$; ρ je velikost vektoru od počátku po pixel (x, y); θ je úhel mezi vektorem ρ a osou x v proti směru hodinových ručiček.

Všimněte si, že podmínka ze Zernikových momentů „$n - |m|$" byla vypuštěna. Proto pro n-tý stupeň obsahuje sada Pseudo-Zernikových momentů asi dvakrát více polynomů než obecné Zernikovy momenty. Kvůli tomu zachytí obnovený obraz pomocí Pseudo-Zernikových momentů více detailů než pomocí obecných Zernikových momentů stejného stupně. Teh a Chin také prokázali, že Pseudo-Zernikovy momenty jsou méně citlivé na šum v obraze než obecné Zernikovy momenty. [5]

4. Implementace nástroje

Uživatelská příručka se nachází na přiloženém CD. Přiložené CD dále obsahuje také kompletní zdrojové kódy celé aplikace se vzorovým projektem pro rozpoznávání tištěného textu. Pro správný běh aplikace je nutné její spuštění v operačním systému Microsoft Windows XP a vyšší, s nainstalovaným .NET Frameworkem 3.5. Veškeré soubory pro instalaci potřebných součástí pro správný běh aplikace naleznete na přiloženém CD.

4.1 Softwarové vybavení

Pro samotnou implementaci byl zvolen programovací jazyk C# ve verzi 3. Dále je využito jazyka XML pro ukládání potřebných aplikačních proměnných a pro udržování informací o projektech. Jako implementační prostředí jsem zvolil Microsoft Visual Studio 2008. Tento software je mimo jiné určen pro vývoj aplikací v jazyce C# a nabízí proto spoustu dodatečných sofistikovaných nástrojů. Pro různé grafické úpravy jsem zvolil software Adobe Photoshop CS3 v trial verzi. K sestavení uživatelské dokumentace bylo použito softwaru od firmy Microsoft. Jako textový editor byl zvolen Microsoft Word 2007, jako tabulkový procesor Microsoft Excel 2007 a pro tvorbu diagramů Microsoft Visio 2007.

4.2 Hardwarové vybavení

Implementace bude probíhat na stroji:
Apple MacBook Pro 13“, Intel Core 2 Duo 2,26GHz, 2GB Ram, 160GB Disk

Jelikož se jedná o aplikaci pro operační systém Microsoft Windows, je potřeba provádět implementaci ve virtuálním stroji VirtualBox, ve kterém je jako operační systém nainstalován Microsoft Windows 7 Professional.

Toto hardwarové vybavení stále ještě postačuje pro pohodlný vývoj této aplikace.

Požadavky na běh aplikace:
Jedná se o desktopovou aplikaci, která poběží v operačním systému Microsoft Windows XP a vyšším, s minimální operační pamětí počítače 1GB Ram. Ovšem pro bezproblémový běh aplikace doporučuji mít operační paměť vyšší.

4.3 Použité algoritmy pro řešení vad a nedokonalostí vstupních materiálů

Hlavní důraz při implementaci nástroje byl kladen problematice úpravy vstupního materiálu. V této části je využito velké množství algoritmů k úpravě obrazu, pro získání co možná nejkvalitnějšího vstupního obrazu, použitého dále v závěrečné OCR části. Následuje popis implementace a zhodnocení použitých algoritmů, v takovém pořadí, v jakém by se mělo při úpravě vstupního obrazu postupovat. Kde je to možné, tak je prováděno zhodnocení mého řešení s řešením volně dostupného nástroje AForge .Net, ve verzi 2.1.0, který disponuje některými funkcemi pro úpravu obrazu.

Jak bylo zmíněno výše, veškeré zdrojové soubory se nacházejí na přiloženém CD a tak je v textu nebudu uvádět.

4.3.1 Uchovávání barevného obrazu

Pro uchovávání barevného obrazu jsem volil mezi možností uchování obrazu v proměnné typu Bitmap nebo v poli bajtů. Zvolení bajtových hodnot pro pole má své důvody. Protože každou intenzitu jasu lze vyjádřit v rozmezí od 0 do 255 a právě proto se volí bajtové hodnoty. U proměnné typu Bitmap není potřeba žádné další úpravy pro uložení vstupního obrazu, na druhou stranu u proměnné typu pole je nutné provádět přepočet při převodu ze vstupního obrazu do pole a naopak.

Při dalším zkoumání a testování jsem se rozhodl pro použití druhé varianty a tedy varianty s ukládáním informací obrazu do pole bajtů. Toto pole je třírozměrné, kde první dva rozměry představují x-ovou a y-ovou pozici pixelu a poslední rozměr představuje postupně intenzitu červené, zelené a modré barvy v aktuálním pixelu.

Díky uchování informací o obrazu v proměnné typu pole je dále jednodušší a rychlejší provádění potřebných algoritmů pro úpravu obrazu.

Samozřejmostí je také obrácený převod, takže převod z proměnné typu pole do proměnné typu Bitmap. Kde probíhá převod načtením všech potřebných hodnot RGB modelu pro každý pixel z pole a následně uložení této hodnoty do příslušného pixelu proměnné Bitmap. Díky této možnosti je jednoduché uložit v jakékoliv fázi úprav obraz do souboru jpg.

Důrazně doporučuji nastavit v aplikaci pro každý obraz originální barevnou kopii, díky které můžete později určovat, zda je vybraná část blok s textem nebo obrázek. Tato volba by měla být použita vždy před volbou převodu obrazu na odstíny šedé barvy, takže při zarovnání obrazu nebo při otočení obrazu podle Vámi zadaného úhlu.

4.3.2 Převod barevného obrazu RGB na odstíny šedé

Jelikož pro ukládání barevného obrazu bylo použito trojrozměrného pole, tak nebude žádný problém pro převod tohoto obrazu na odstíny šedé. U barevného obrazu bylo nutnost mít právě trojrozměrné pole, protože jsme potřebovali uložit pro každý pixel všechny tři barevné složky.

U šedotónového obrazu už tyto tři složky ukládat nemusíme a tak můžeme použít dvourozměrné pole, ve kterém bude pro každou x-ovou a y-ovou hodnotu uložena intenzita jasu pro aktuální pixel obrazu. Každá složka RGB modelu se tedy propočítá podle vzorce [Vzorec 1.] a výsledná intenzita se uloží do proměnné. U dalších operací se pak využívá této proměnné, protože se operace provádějí na šedotónovém obraze. Výsledek použité metody pro převod barevného obrazu můžete vidět na obrázku [Obrázek 7.].

Pro převod barevného RGB obrazu na odstíny šedé se může použít třídu GrayScale z nástroje AForge .NET. Ovšem zde se objevuje první problém, kterým je nutnost jako vstupního argumentu použít obraz v proměnné typu Image. Z důvodu uložení obrazu v proměnné typu pole by nutná konverze z pole na Image a poté konverze z Image na pole byla zbytečná a u pomalejších počítačů by byla časově znatelná.

Výsledek metody ApplyInPlace, třídy GrayScale z nástroje AForge .NET není u velkých obrazů tak kvalitní jako výsledek při výpočtu jasové hodnoty z proměnné typu pole. I z tohoto důvodu jsem od použití této metody upustil a zařadil do výsledné aplikace můj algoritmus pro převod barevného obrazu RGB modelu do obrazu v odstínech šedé.

4.3.3 Geometrické transformace

Tyto transformace mohou být použity jak na barevný obraz, tak na šedotónový obraz. V aplikaci se využívá pouze otočení a změna měřítka. Posunutí není v případě úpravy obrázku potřeba a zkosení také ne. Pokud provádíte otočení několikrát za sebou, mohlo by dojít k deformaci a rozmazání obrazu. Proto se v aplikaci provádí otočení vždy proti originálnímu obrazu. Tím je zajištěno, že se bude otáčet vždy originálním obrazem.

Změna měřítka se vypočítává vždy z aktuálně zobrazeného obrazu. Takže není nutné ukládat originální obraz z důvodu kvalitního výpočtu. Originální obraz se v tomto případě ukládá pro možnost navrácení obrazu do původního měřítka. Pokud ovšem nepotřebujete vybrat pro další potřeby pouze část obrazu, je změna měřítka nepotřebná, protože pozdější algoritmy pro rozpoznání řádek textů v obraze naleznou všechny řádky s nastavenou minimální výškou.

Tyto transformace se propočítávají podle maticových transformací. V tomto případě jsem zvažoval možnosti provádět operace vlastními metodami, protože když je obraz uložen v proměnné typu pole, tak dopočítat transformaci není tak velký problém. Právě díku uložení obrazu do proměnné typu pole, což je vlastně matice, se mohou propočítat nové souřadnice pixelů například pro otočení podle vzorce [Vzorec 5.]. Obdobně by se vypočítaly nové souřadnice pro změnu měřítka, ovšem podle patřičného vzorce pro takový výpočet.

V tomto případě jsem ale zvolil třídy z nástroje AForge .NET, přesněji RotateBicubic, respektive Scale. Tyto třídy sice opět pracují s obrazem, uloženým v proměnné typu Image, ale výsledek je podstatně lepší, než výsledek z mé metody. Třída Scale se postará o potřebné zvětšení, případně zmenšení, obrazového plátna. Výsledek rotace obrazu můžete vidět na obrázku [Obrázek 8.].

Později jsem byl nucen doplnit aplikaci o svou vlastní metodu pro změnu měřítka, protože pokud se na obraze vyskytoval jeden nebo více objektů pro výběr relevantní oblasti, musely se jeho nové souřadnice také dopočítat. Z tohoto důvodu jsem musel dopsat metodu, která dopočítává nové

souřadnice pro tyto objekty. Algoritmus pracuje se souřadnicemi každého objektu a následným propočítáním nových souřadnic pomocí maticových transformací. Při změně měřítka obrazu je zkontrolováno, jestli se na obraze nachází objekty pro výběr relevantních oblastí. Pokud ano, jsou získány souřadnice všech těchto objektů, dále hodnota změny měřítka a nakonec se provede potřebný propočet.

4.3.4 Inverze barev

Metoda pro inverzi barev pracuje jak s barevným obrazem, tak s šedotónovým obrazem. Ovšem u problematiky úpravy vstupního obrazu pro fázi OCR se inverze barevného obrazu moc nepoužívá. U šedotónového obrazu je tato metoda použitelnější. Pokud je například bílý text v černém poli, tak se při aplikaci inverze tyto barvy zamění, takže bude černý text v bílém poli. Což je pro pozdější použití přijatelnější. Tato metoda lze v programu aplikovat buď na celý obraz nebo na jeho vybranou část, která bude popsána později. Získání obrazu s invertovanými barvami je provedeno průchodem přes všechny hodnoty v poli, ve kterém je uložen vstupní obraz, a spočítáním invertované hodnoty pro každou hodnotu z výběru pole. Metoda inverze barev vrací upravené pole, které se dále převede do proměnné typu Bitmap a ta je nakonec prezentována uživateli. Obraz po provedení inverze barev můžete vidět na obrázku [Obrázek 9.]. Tato metoda je poměrně jednoduchá a rychlá tak jsem ji ani neporovnával s třídou Invert z nástroje AForge .NET.

4.3.5 Gama korekce

Tato metoda se převážně využívá při úpravě špatně nasvíceného obrazu. Takže pokud je Váš vstupní obraz špatně oskenovaný, pak je nejvhodnější použít tuto metodu. Její využití je už pouze u šedotónového obrazu. Opět je aplikace této metody volitelná mezi celým obrazem a výběrem obrazu. Aplikace této korekce se provádí průchodem přes každou hodnotu pole a spočítáním minimální intenzity jasu v pixelu. Tato intenzita se spočítá jako minimální hodnota ze 3 hodnot. Tou první je hodnota 255, která představuje černou barvu a těmi ostatními jsou poměr hodnoty z aktuálního pixelu vůči hodnotě 255 a invertovaný stanovený koeficient vynásobený hodnotu 255. Pokud byl vstupní obraz příliš tmavý nebo příliš světlý, pak je po aplikaci metody gama korekce obraz použitelnější a lépe čitelný. Tato metoda opět vrací upravené dvourozměrné pole, které je převedeno na proměnnou typu Bitmap. Po provedení operace gama korekce získáte výsledný obraz, který můžete videt na obrázku [Obrázek 10.].

Nástroj AForge .NET nabízí třídu pro gama korekci. Tato třída se jmenuje GammaCorrection, u které se jako jediný parametr udává právě koeficient. Tuto třídu jsem pouze otestoval, co se týče kvality výstupu. Dále jsem s ní nepracoval, protože poskytuje korekci pouze celého obrazu. Takže proto by nešla aplikovat pouze na určitou část obrazu, což je v tomto případě velký nedostatek.

4.3.6 Logaritmická transformace

V některých případech je šedotónový obraz příliš kontrastní. Hlavně pokud je tento obraz nekvalitně oskenovaný z časopisu a obsahuje obrázky, ve kterých je text. Primárním úkolem je tedy získat tento text a obrázky potlačit do pozadí. Právě k tomu se může použít logaritmická transformace, která dokáže vysoké hodnoty jasu v obraze tzv. „zkomprimovat“. Opět se může aplikovat buď na celý obraz nebo na jeho vybranou část. Základem pro celý výpočet je získání hodnoty maximální intenzity jasu. Tato hodnota je tedy vypočítána na začátku algoritmu a dále se s ní pracuje u výpočtu konstanty pro logaritmický výpočet. Tato konstanta je spočítána poměrem hodnoty 255 vůči destítkovému logaritmu z hodnoty maximální intenzity. Tím je získáno všech potřebných pomocných hodnot pro spočítání nové intenzity v každém pixelu obrazu. Nová intenzita je nakonec spočítána vynásobením konstanty pro logaritmický výpočet s desítkovým logaritmem pro každou intenzitu jasu ze vstupního pole. Opět je vráceno dvourozměrné pole, které je převedeno na proměnnou typu Bitmap a prezentováno uživateli. Výsledek této metody je na obrázku [Obrázek 11.].

Třída pro provádění logaritmické transformace není v nástroji AForge .NET vůbec zahrnuta. Takže jsem neměl možnost provádět jakékoliv zhodnocení.

4.3.7 Roztažení kontrastu

Roztažení kontrastu se může použít v opačných případech, než se používala logaritmická transformace. Pokud budeme chtít rozpoznávat například starý text, napsaný na psacím stroji, pak je dost možné, že bude takový text zašlý a špatně rozpoznatelný. Ovšem při použití metody roztažení kontrastu bude zašlý text tzv. „roztažen“ vůči pozadí. Tato metoda je stejně jako předchozí metody aplikovatelná buď na celý obraz nebo na jeho vybranou část. Zde se ale doporučuje její použití na celý obraz, protože tak bude získáno potřebných informací o celém obraze. V opačném případě by mohly vzniknout některé faktory, které by pozdější rozpoznávání znepříjemňovaly. U této metody je na začátku spočítána minimální a maximální hodnota jasu. Tyto dvě hodnoty jsou dále používány pro výpočet koeficientu změny každého pixelu v obraze. Koeficient pro každý pixel je získán součinem rozdílu aktuální hodnoty jasu pixelu s minimální hodnotou jasu a podílu hodnoty 255 s rozdílem maximální a minimální hodnoty jasu. Celá tato hodnota je pak nakonec vydělena hodnotou intenzity jasu pro aktuální pixel. Tímto výpočtem je získáno koeficientu změny pro každý pixel v obraze. Teď už nic nebrání výpočtu nové intenzity jasu pro každý pixel, která je spočítána jako součin aktuální intenzity jasu pixelu s aktuálním koeficientem změny pro pixel. Návratovou hodnotou metody roztažení kontrastu je upravené dvourozměrné pole. Takže pro jeho prezentaci uživateli je nutné toto pole nakonec převést do proměnné typu Bitmap. Výsledek metody roztažení kontrastu můžete vidět na obrázku [Obrázek 12.], ze kterého je patrné, že při aplikaci roztažení kontrastu na celý obraz získáme lepší výsledek, než kdybychom tuto metodu aplikovali pouze na určitou část obrazu. Pokud by totiž byla metoda aplikovaná jen na určitou část obrazu, vznikl by nevhodný obraz s různě roztaženým kontrastem.

Nástroj AForge .NET obsahuje třídu pro provádění roztažení kontrastu. Tato třída se nazývá ContrastStretch. Výsledkem je obraz podobný, obrazu z mé metody. Ovšem v tomto případě je

nevýhodou opět vstupní parametr pro inicializaci třídy. Tím je proměnná typu Image, a tak provádět převod šedotónového pole do proměnné typu Image a poté tuto proměnnou předat konstruktoru třídy ContrastStretch, dále pak na tuto proměnnou aplikovat filtr a znovu ji převést do dvourozměrného pole je zbytečně zdlouhavé. Takže jsem od zařazení této třídy do mé aplikace upustil.

4.3.8 Histogram

Histogram je vypočítaný pouze pro šedotónový obraz. Výpočtem histogramu je získáno informativních hodnot pro pozdější určování prahové hodnoty. Získání potřebných hodnot je provedeno procházením celého dvourozměrného pole obrazu a ukládáním patřičných hodnot do proměnné typu pole, která slouží pro uložení hodnot každého stejného jasu v obraze. Tyto hodnoty jsou nakonec prezentovány uživateli v podobě grafu. K vykreslení grafu jsem použil volně dostupnou komponentu společnosti Microsoft, Microsoft Chart fot Windows Forms. Pokud je k dispozici histogram obrazu, může být pro Vás podstatně jednodušší určit prahovou hodnotu obrazu, která bude vysvětlena později.

Histogram se počítá ve všech aplikacích stejně. Nástroj AForge .NET proto nenabízí žádnou metodu pro získání hodnot obrazu do histogramu. Z tohoto důvodu jsem tedy pro výpočet hodnot histogramu použil vzorce [Vzorec 8.]. Výsledný histogram tak, jak ho vykreslí komponenta Microsoft Chart for Windows Forms, můžete vidět na obrázku [Obrázek 13.].

4.3.9 Vyhlazování průměrováním

Nejprve musím popsat metodu pro aplikaci masky. Metoda aplikace masky přijímá 5 argumentů. První a druhý argument je x-ová a y-ová souřadnice pixelu, na který se má aplikovat maska, třetím argumentem je šedotónové pole, čtvrtým argumentem je maska, takže pole, která se aplikuje na pixel a posledním argumentem je dělitel, kterým se upraví každá hodnota masky. Pokud je hodnota argumentu dělitele určena jako 0, pak se hned na začátku upraví na hodnotu 1. Dále se zjistí středová souřadnice předané masky a suma pro aktuální pixel, která se počítá vždy ze všech hodnot masky. Díky výpočtu celkové sumy se na tuto hodnotu následně aplikuje podíl s předávaným dělitelem a tím je získána poslední potřebná proměnná s názvem result. Hodnota, kterou vrátí algoritmus pro aplikaci masky je nakonec spočítána jako maximální hodnota z hodnoty 0, to pokud by byl počáteční pixel nulový, a z nižší dvojice hodnot 255 nebo hodnoty proměnné result.

U metody vyhlazování průměrováním se používá maska [Vzorec 10.]. Tato maska je již upravená dělitelem. Maska se předává vždy při procházení celým šedotónovým polem pro aktuální pixel. Metoda lze opět aplikovat buď na celý obraz nebo na jeho vybranou část. Návratovým typem metody vyhlazování průměrováním je opět dvourozměrné pole. Výsledné upravené pole je nakonec převedeno do proměnné typu Bitmap a ta je prezentována uživateli.

Nástroj AForge .NET nabízí třídu pro vyhlazování průměrováním. Přesněji se jedná o třídu Mean. U této třídy je pevně stanovena hodnota proměnné dělitel na hodnotu 9. Tuto metodu jsem zvažoval zařadit do mé aplikace, ale opět mě od této volby odradila nemožnost výběru oblasti pro provádění

vyhlazování. U některých metod by absence volby oblasti tak moc nevadila, ale u vyhlazování je to velkým nedostatkem.

4.3.10 Mediánová filtrace

U mé metody pro mediánovou filtraci se pro posun po obraze používá maska o velikosti 3 x 3. Takže se z každé části obrazu o velikosti 3 x 3 vybere 5. prvek, který představuje medián aktuální masky. Toho je docíleno průchodem přes celý obraz, nebo v případě vybrání části obrazu pouze přes vybranou část. V každém průchodu se do masky uloží potřebné hodnoty z okolí aktuálního pixelu, tak aby měla maska velikost 3 x 3. Následně se hodnoty v masce seřadí podle velikosti a vybere se 5. prvek, který se uloží do dvourozměrného pole s informacemi o obrazu na adresu aktuálního pixelu. Tím je v podstatě zajištěno odstranění nežádoucího šumu z obrazu, tak jak je to vidět na obrázku [Obrázek 15.]. Z tohoto důvodu je tato metoda nejvyužitelnější v případě špatně naskenovaného vstupního obrazu. Návratovým typem metody je dvourozměrné pole, které je převedeno do proměnné typu Bitmap. Ve většině případů je doporučeno použít právě tuto metodu pro odstranění nežádoucího šumu, který by mohl negativně ovlivnit kvalitu obrazu při provedení operace prahování, což vysvětlím později. Jak bylo zmíněno, může být metoda aplikována také na vybranou část obrazu, což je přínosem pro rozsáhle obrazy, na kterých se šum vyskytuje jen na některých místech, takže by bylo zbytečné aplikovat mediánovou filtraci na celý obraz. Tato metoda vrací dvourozměrné pole, které je pro prezentaci výsledku uživateli ještě převedeno na proměnnou typu Bitmap. Výsledek mediánové filtrace můžete vidět na obrázku [Obrázek 15.]. Na tomto obrázku můžete také vidět kvalitní odstranění nežádoucího šumu z obrazu.

Nástroj AForge .NET nabízí pro mediánovou filtraci třídu Median. Tato třída pracuje s maskou o velikosti 5 x 5, takže vybírá z masky 13. prvek, který představuje medián. Provádění mediánové filtrace je v porovnání s mým řešením pomalejší, a to z důvodu vstupního argumentu pro konstruktor třídy. Tím je opět proměnná typu Image, takže při pohybu masky po obraze je prováděno nejprve propočítání hodnot pro uložení do masky a nalezení hodnoty mediánu v této masce. Maska je větších rozměrů než u mého řešení a tak je trvání této operace o zlomek delší než v mém případě. Absence možnosti určení oblasti pro provádění mediánové filtrace je opět hlavním nedostatkem tohoto nástroje.

4.3.11 Gaussovo vyhlazování

U metody gaussova vyhlazování je použita opět aplikace masky, jako tomu byl v případě metody vyhlazování průměrováním. Zde je ale použita maska o velikost 5 x 5 a dělitel s hodnotou 115. Při použití gaussova vyhlazování je výsledný obraz spíše rozmazaný, takže se používá v případech příliš ostrého obrazu. Opět je možnost výběru celého obrazu nebo pouze jeho určité části. Zde je výběr oblasti velkým přínosem například u přílišné ostrosti ve střední části obrazu. Takže nedojde k nežádoucímu rozmazání u části obrazu, u které toho nechceme dosáhnout. Použitelnost této metody je u velice ostře oskenovaných obrazů nebo u nekvalitně vyfotografovaného obrazu, který je pro použití v aplikaci oskenován sice kvalitně, ale z nekvalitního zdroje. Návratovým typem metody je

opět dvourozměrné pole, které musí být pro případ prezentování výsledku uživateli převedeno do proměnné typu Bitmap. Výsledek použití metody gaussova vyhlazování je na obrázku [Obrázek 17.], na kterém můžete vidět také nežádoucí rozostření, které se na obrázku objevilo při použití metody na celý obraz.

Nástroj AFogre .NET nabízí třídu pro gaussovo vyhlazování. Tato třída se nazývá GaussianBlur a jako vstupní argumenty přebírá hodnotu velikosti středu obrazu a hodnotu Gaussovy křivky. Zde je pozitivní vlastností určení hodnoty Gaussovy křivky, což může být v některých případech výhodou pro výsledný obraz. Ovšem opět je nevýhodou pro metodu nutnost přepočítat ze vstupní proměnné typu Image hodnoty pro výpočet gaussova vyhlazování a také aplikace vyhlazování na celý obraz, což může způsobit nežádoucí rozmazání.

4.3.12 Sobelův operátor

Sobelův operátor pro detekci hran byl do aplikace přidán hlavně z důvodu získání textu na barevném pozadí. Například pokud je text zapsán v obrázku nebo pokud je v barevném poli. Takový text by bez potřebných úprav v dalších částech zanikl a tím by bylo výsledné rozpoznávání vyhodnoceno jako nekvalitní. V aplikaci je možno Sobelův operátor použít ve třech variantách. Ve směru osy x, ve směru osy y nebo pod úhlem $45°$. Nejlepší výsledky při získání textu z barevného pozadí jsou při použití metody ve variantě pod úhlem $45°$.

V aplikaci je použita pro výpočet Sobelova operátoru maska o velikosti 3 x 3 a pro každou variantu je použita různá maska. Algoritmus začíná výpočtem potřebného součtu gradientu a uložením koeficientu do potřebného pole s hodnotami gradientu pro aktuální pixel. Tento koeficient je spočítán pro aktuální pixel jako součin každé hodnoty z masky s příslušnou hodnotou pixelů, které leží pod maskou. Tato hodnota koeficientu je přičtena k předchozímu součtu gradientu. Po ukončení výpočtu součtu gradientu je tato hodnota použita pro výpočet prahové hodnoty, která je vypočítána podílem součtu hodnot a šířkou obrazu. Nakonec se již vypočítávají hodnoty barvy pro každý pixel. Kdy se porovnává hodnota gradientu aktuálního pixelu s prahovou hodnotou. Pokud je prahová hodnota menší nebo rovna s hodnotou gradientu aktuálního pixelu, pak je tato barva prohlášena za bílou a pokud je prahová hodnota nižší než hodnota gradientu aktuálního pixelu, pak je tato barva prohlášena za černou. Návratovým typem mé metody pro Sobelův operátor je dvourozměrné pole, které je pro potřebu prezentace uživateli převedeno do proměnné typu Bitmap. Výsledek použití Sobelova operátoru ve variantě při rotaci o $45°$ můžete vidět na obrázku [Obrázek 19.]. Všimněte si kvalitního získání textu z barevně ohraničeného obrazu, takže v případě obrazu vepsaného do barevného rámečku nebo obrazu získáme kvalitní výsledek pro další použití. Toto je také jeden z důvodů, proč byla metoda pro detekci hran pomocí Sobelova operátoru zařazena do aplikace.

Nástroj AForge .NET používal v předchozích verzích zvlášť třídy pro různé hranové detektory. Ovšem od mnou testované verze již používá třídu s názvem Edges, u které si již nelze zvolit použitý operátor pro detekci hran. Z tohoto důvodu jsem do aplikace zahrnul mou metodu pro Sobelův operátor, protože nástroj AForge .NET by dokázal jen těžko odhadnout, že chce uživatel použít právě Sobelův operátor pro získání textu z barevného pozadí.

4.3.13 Operátor Prewittové

Detekce hran pomocí operátoru Prewittové je v aplikaci zařazen hlavně z důvodu kvalitních výsledků při detekci hran u textu napsaném na minci. Operátor Prewittové je možno použít ve variantách ve směru osy x nebo ve směru osy y. Pro případ textu na minci jsou kvalitnější výsledky při použití varianty ve směru osy y.

V aplikaci je opět použita maska o velikosti 3 x 3, pro každý směr s jinými hodnotami. Algoritmus pracuje obdobně jako algoritmus u Sobelova operátoru, dosahuje ovšem jiných výsledku z důvodu hodnot, použitých v masce.

Pokud tedy budete používat aplikaci pro detekci textu na minci, jsou výsledky pro detekci hran pomocí operátoru Prewittové ze všech metody, které nabízí aplikace, nejkvalitnější. Navrácené dvourozměrné pole musí být nakonec ještě převedeno do proměnné typu Bitmap, aby mohl být výsledek metody prezentován uživateli. Výsledek použití metody pro detekci hran pomocí operátoru Prewittové můžete vidět na obrázku [Obrázek 18.].

U nástroje AForge .NET je problém stejný jako v předchozí kapitole. Takže není možno vybrat přesně typ operátoru, který chceme pro detekci hran použít.

4.3.14 Laplacian of Gaussian

Metoda Laplacian of Gaussian je v aplikaci zařazena pro vyplnění mezery mezi předchozími operátory pro detekci hran. Tuto metodu je nejvhodnější použít u nekvalitních obrazů, které mohou mít velké množství šumu, zabraňující předchozím operátorům provádět kvalitní detekci hran. Ovšem nevýhodou tohoto operátoru je v některých případech nežádoucí rozmazání obrazu, což je způsobeno Gaussovým operátorem. Právě z tohoto důvodu je doporučeno aplikovat tuto metodu pro detekci hran na nekvalitně oskenovaný obraz.

Metoda Laplacian of Gaussian využívá metodu pro aplikaci masky, podobnou z metody pro vyhlazování. Ovšem s tím rozdílem, že tato metoda nepřebírá 5. argument, dělitel. Metoda pro aplikaci masky u Laplacian of Gaussian přebírá tedy argumenty 4. První dva argumenty jsou x-ová a y-ová souřadnice aktuálního pixelu, třetím argumentem je celé šedotónové pole a posledním argumentem je předávaná maska, která je v tomto případě ve velikosti 5 x 5. Na začátku metody pro aplikaci masky se získá hodnot prostředního prvku předávané masky, který je dále používán pro výpočet hodnoty sumy. Hodnota sumy se počítá procházením celé předávané masky a následným součtem původní hodnoty sumy se součinem hodnoty zvoleného pixelu předávaného šedotónového pole s aktuální hodnotou předávané masky. Zvolený pixel předávaného šedotónového pole se získá pro osu x rozdílem předávané x-ové souřadnice s prostřední hodnotou masky a následným součtem s aktuální hodnotou procházeného řádku v cyklu. Obdobně je spočítán také zvolený pixel pro osu y. Pouze s tím rozdílem, že se konečný součet provádí s aktuální hodnotou procházeného sloupce v cyklu. Hodnota sumy je nakonec porovnávána při rozhodování, zda má metoda aplikace masky vrátit hodnotu 0 nebo 255.

Tímto způsobem je získána hodnota pro každý pixel z šedotónového obrazu, čehož je dosaženo průchodem celé přes dvourozměrné pole, které představuje šedotónový obraz. Návratovým typem

metody pro výpočet Laplacian of Gaussian je dvourozměrné pole. Takže pro prezentaci uživateli musí být toto pole ještě převedeno do proměnné typu Bitmap. Výsledek této metody můžete vidět na obrázku [Obrázek 21.], zde by bylo potřeba použít na obrázek metodu pro odstranění šumu. V našem případě se jako vhodná metoda nabízí Mediánová filtrace.

Nástroj AForge .NET opět neobsahuje třídu pro výpočet Laplacian of Gaussian, takže jsem nemohl mou metodu porovnat s tímto nástrojem.

4.3.15 Globální prahování

Metoda globálního prahování je v mé aplikaci použita dvakrát. První použití je zde z důvodů standardního provádění globálního prahování a druhé použití je zde z důvodu menších problémů v prvním případě.

První použití metody globálního prahování pracuje s třídou Threshold z nástroje AForge .NET. Prahování v případě tohoto nástroje je poměrně rychlé a kvalitní. Volbu nástoje AForge .NET pro globální prahování můžu zdůvodnit rychlejším prováděním a také možností uživatele si dynamicky měnit hodnotu prahu. Pokud bych pro globální prahování chtěl použít mou metodu, pak bych musel metodě předávat šedotónové pole, u kterého by se při každé změně hodnoty prahu provádělo prahování. To by vedlo k příliš velké prodlevě před získáním potřebného výsledku. Pokud jsem ale pro globální prahování použil třídu Threshold z nástroje AForge .NET, stačilo pouze před prvním použitím převést šedotónoté pole do proměnné typu Image a tu pak předávat třídě Threshold. V takovém případě byla prodleva minimální. Po potvrzení prahové hodnoty a tím tedy provedení prahování stačilo pouze převést obraz z proměnné typu Image zpět do dvourozměrného pole, aby se s ním dalo dále pracovat. Pro zjednodušené určení prahové hodnoty je v aplikaci použito vykreslení histogramu pomocí komponenty Microsoft Chart fot Windows Forms, což bylo zmíněno výše.

Druhé řešení v aplikaci je zde z důvodu použití prahování u obrazu s mincí. Pokud jsem totiž použil třídu Threshold na obraz s mincí, kde je text vlastně pokryt na některých místech stínem, vrženým hranou mince, pak byl výsledný prahovaný obraz znehodnocen právě stínem z hrany. Takže bylo dosaženo pouze toho, že se text z mince ztratil v černé barvě. Proto jsem byl nucen do aplikace přidat druhou metodu pro provádění globálního prahování. U této metody se počítá s tím, že si uživatel zvolí hodnotu prahu takovou, aby byl po prahování vidět hlavně text na minci. Pokud byla totiž mince správně oskenována, pak mají hrany textu větší jasovou hodnotu než stín, vržený hranou mince. V takovém případě je po určení prahové hodnoty nežádoucí stín odstraněn a potřebný text je od nežádoucího stínu oddělen. Takového výsledku je ovšem dosaženo pouze v případě kvalitně oskenované mince. Výsledek metody globálního prahování můžete vidět na obrázku [Obrázek 22.]. Z obrázku je patrné, že použití metody globálního prahování pro získání kvalitních výsledků je velice diskutabilní. Pokud bychom tedy chtěli použít pro segmentaci obrazu právě metodu globálního prahování, musel by být vstupní obraz opravdu kvalitní, s minimálním množstvím šumu a s minimálním množstvím obrázků. Z toho vyplývá, že by musel vstupní obraz obsahovat pokud možno pouze text. V takovém případě by byl výsledný obraz použitelný pro další práci a tím pádem by nebylo potřeba globálně prahovaný obraz ještě dále upravovat.

4.3.16 Prahování Otsu

Prahování metodou Otsu je v aplikaci z jednoho hlavního důvodu. Tímto důvodem je kvalitní automatické nalezení prahové hodnoty, čehož je v aplikaci využito při hromadném zpracování textu. Algoritmus Otsu prahování je poměrně rychlý, ale v případě velkého množství vstupních obrazů se musí počítat s určitou časovou prodlevou.

Na začátku celého algoritmu se vypočítají hodnoty histogramu. Tento histogram se musí dále normovat. Toho je dosaženo průchodem všech hodnot původního histogramu a jejich podíl s hodnotou všech pixelů v šedotónovém poli. Takto normovaný histogram je potřeba dále kumulovat, takže je prováděna operace kumulace histogramu a to tím způsobem, že první hodnota nového kumulovaného histogramu bude totožná s hodnotou normovaného histogramu. A každá další hodnota je součtem hodnoty aktuálního hodnoty normovaného histogramu s předchozí hodnotou normovaného histogramu. Nakonec jsou spočítány potřebné koeficienty pro pomocný výpočet určení maximální hodnoty rozptylu, která se určuje průchodem přes kumulovaný a normovaný histogram. Po ukončení průchodu těchto dvou histogramů je nalezena hodnota maximálního rozptylu. Společně s hodnotou maximálního rozptylu je také nalezena adresa této hodnoty v poli s histogramem. Takže je vlastně tato adresa stejná jako adresa hodnoty maximálního rozptylu. Když už známe potřebnou adresu, pak už nám nic nebrání výpočtu Otsu prahování. Pro prahovou hodnotu je totiž použita nalezená adresa maximálního rozptylu. Metoda, pro výpočet Otsu prahování vrací dvourozměrné pole, které musí být pro prezentaci uživateli převedeno do proměnné typu Bitmap. Výsledek mé metody pro výpočet Otsu prahování můžete vidět na obrázku [Obrázek 23.]. Zde je vidět kvalitně segmentovaný obraz, který je vhodný pro další použití. Z tohoto důvodu byla metoda Otsu prahování zařazena pro prahování obrazu při použití hromadného zpracování obrazu.

Nástroj AForge .NET obsahuje třídu pro Otsu prahování. Třída se nazývá OtsuThreshold a její výhodou je to, že vrátí kvalitní výsledek za poměrně krátkou dobu. Ovšem za nutnosti předávat konstruktoru třídy opět obraz uložený v proměnné typu Image. Pro použití u hromadného zpracování textu by to tak velkou nevýhodou nebylo, protože v takovém případě čeká uživatel na výsledný převod a vlastně do běhu zpracování nezasahuje. Takže je výhodou mít obraz uložený v proměnné, ze které lze obraz dále uložit do souboru, ale v případě kdy si uživatel zvolí jako metodu prahování metodu Otsu se musí předávat pole s šedotónovým obrazem. Takže z tohoto důvodu jsem použil v aplikaci svou metodu.

4.3.17 P-procentní prahování

U metody p-procentního prahování se využívají další 2 metody pro pomocné výpočty. Tou první metodou je metoda pro výpočet histogramu masky a tou druhou metodou je metoda pro výpočet prahové hodnoty.

Metoda pro výpočet histogramu masky přebírá 5 argumentů. Prvním 2 argumenty představují x-ovou a y-ovou souřadnici aktuálního pixelu z dvourozměrného pole, které představuje šedotónový obraz. Dalším argumentem je dvourozměrné pole šedotónového obrazu. Předposlední argument je pro určení, zda se jedná o první sloupec obrazu nebo ne. A posledním argumentem je hodnota delta, která

představuje poloviční velikost použité masky. V tomto případě je maska o velikosti 5, takže hodnota delta bude mít hodnotu 2,5. Výpočet histogramu masky dále pokračuje podle hodnoty předposledního argumentu. Pokud se jedná o první sloupec obrazu, jsou hodnoty histogramu ukládány ve dvou vnořených cyklech, ale pokud se jedná o jiný sloupec, pak jsou hodnoty ukládány nejprve s jednou pevně danou hodnotou x-ové souřadnice a poté s jinou pevně danou hodnotou stejné souřadnice. Celý algoritmus je pro slovní vysvětlení poměrně složitý a tak bude nejlepší se na jeho konstrukci podívat přímo do zdrojového souboru s algoritmy pro úpravu obrazu.

Metoda pro výpočet prahové hodnoty již tak složitá není. Vstupním argumentem je zde hodnota, představující celkový počet pixelů v masce. Pokud tedy byla na začátku velikost masky stanovena na hodnotu 5, pak je celkový počet pixelů 25. Posledním argumentem je procentuální hodnota pro výpočet prahu. Následuje průchod přes celý vypočítaný histogram a sečtení získané hodnoty histogramu s předchozí hodnotou histogramu tak dlouho, dokud nebude výsledná suma větší než součin hodnoty celkového počtu pixelů s procentuální hodnotou. V takovém případě vrátí metoda hodnotu aktuální adresy histogramu. V opačném případě je vrácena hodnota 0.

Nyní jsou popsány obě dvě metody, které se používají při výpočtu p-procentního prahování. Nyní se samotné prahování provádí průchodem přes celé dvourozměrné pole, výpočtem histogramu masky pro aktuální pixel a výpočtem prahové hodnoty. Tato hodnota je porovnána s hodnotou aktuálního pixelu a je rozhodnuto, jestli je nová hodnota bílá nebo černá. Návratovým typem metody pro p-procentní prahování je dvourozměrné pole, které musí být pro prezentaci uživateli převedeno do proměnné typu Bitmap. Výsledek metody můžete vidět na obrázku [Obrázek 24.].

Nástroj AForge .NET nemá žádnou podobnou třídu pro výpočet p-procentního prahování, takže jsem nemohl provádět srovnání mého řešení s řešením AForge .NET.

P-procentní prahování je v aplikaci využito převážně v případě, kdy uživatel zná přibližný procentuální poměr, který zaobírá popředí vůči pozadí. V opačném případě by bylo použití této metody zbytečné, protože by byl výsledek nevhodný pro další použití.

4.3.18 Shrnutí použitých algoritmů pro řešení vad a nedokonalostí vstupních materiálů

V předchozích kapitolách byly shrnuty veškeré použité algoritmy v mé aplikaci, které řeší vady a nedokonalosti vstupních materiálů. Všechny výsledky použitých metod můžete vidět na obrázcích v kapitole [2.2 Řešení vad a nedokonalostí vstupních materiálů]. Použité algoritmy pro řešení těchto problémů jsou kvalitně okomentovány a nacházejí se ve třídě ImageProcessing(), v souboru ImageProcessing.cs na přiloženém disku CD-ROM.

Aplikace slouží také jako jednoduchý editor obrázků, takže je možné po použití jakékoliv výše popsané metody tento obrázek uložit a použít ho v jiné aplikaci. Obrázky z aplikace se mohou ukládat pouze v obrazovém formátu jpg.

4.4 Text na kružnici

Pro úpravu obrazu, který obsahuje prohnutý text, jsem použil co nejjednodušší metodu. Takže nebudu hledat text na kružnici, ale celou kružnici si nejprve narovnám a pak můžu hledat text vlastně stejně, jako kdyby byl na řádku.

Budu tedy provádět úpravu celého obrazu a tudíž nebudu potřebovat pole, ve kterém jsou uloženy informace o obraze. Jedním z důvodu je také možnost provádět tzv. „narovnání“ obrazu kdykoliv. Takže i když je obraz barevný nebo i když je už převedený do odstínů šedé.

Metoda, která provádí celý proces narovnání, přebírá celkem 5 argumentů. Prvním argumentem je výsledná šířka narovnané roviny, která je nejčastěji dvojnásobkem původní šířky vstupního obrazu. Druhým argumentem je obraz, který obsahuje kružnici. Třetím a čtvrtým argumentem jsou x-ová a y-ová souřadnice středu kružnice a posledním argumentem je poloměr kružnice, který vlastně ve výsledku bude představovat výšku narovnané roviny. Je vytvořena nová proměnná, která bude na konci představovat narovnaný obraz. Následuje určení, jestli je vstupní obraz stále barevný nebo už je převeden na odstíny šedé. Dále musí být vstupní obraz a proměnná pro výsledný obraz převedeny na jednorozměrné pole bajtů. Před samotným prováděním narovnání se ještě spočítají poměry stran jak vstupního obrazu, tak budoucího narovnaného obrazu a nyní již nic nebrání tomu, aby bylo zahájeno propočítávání nových souřadnic pro každý pixel z kružnice.

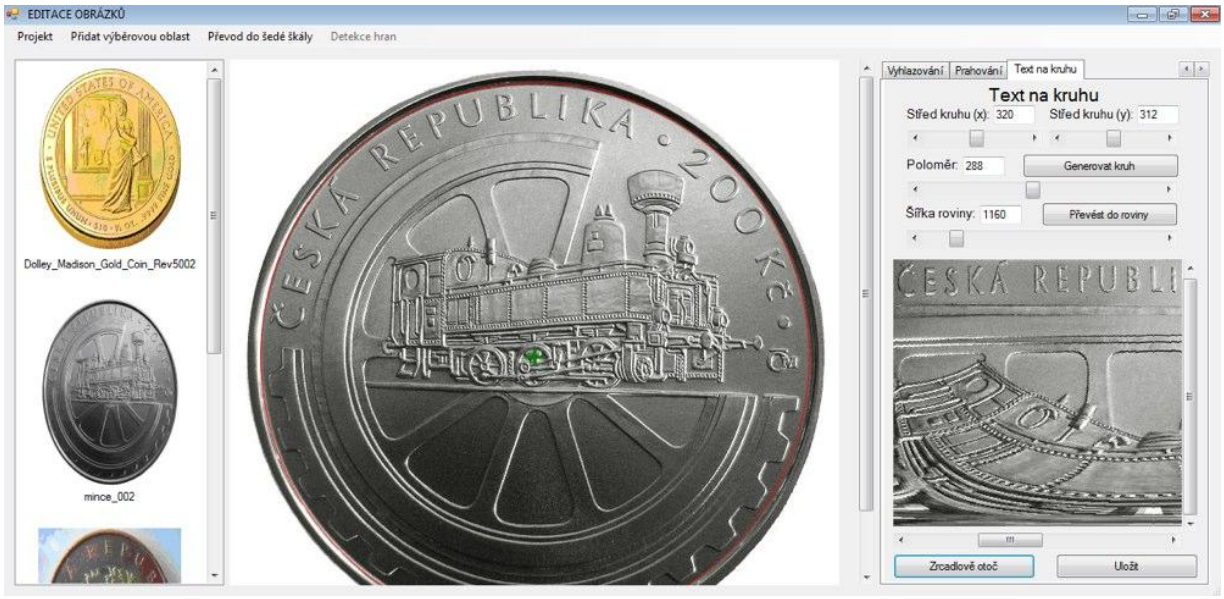
Toto propočítávání je zahájeno průchodem šířkou zdrojového obrazu, ve kterém se vždy spočítá úhel, mezi aktuálním pixelem a středem. Z toho úhlu je spočítána také hodnota sinus a cosinus. Dále se začne procházet výška zdrojového obrazu a použijí se předchozí vypočítané hodnoty pro spočítání nové pozice pixelu. Pokud jsou nově získané pozice z rozsahu, označeném kružnicí, tak se může aktuální pixel zakreslit na novou pozici do proměnné, která představuje narovnanou kružnici. Nakonec je toto jednorozměrné pole převedeno na proměnnou typu Bitmap, která je prezentována uživateli. Může se stát, že podle orientace textu na kružnici, bude výsledný narovnaný obraz zrcadlově otočený. Pro řešení tohoto problému je možno provést ještě zrcadlové otočení obrazu. Celý výsledek této transformace můžete vidět na obrázku [Obrázek 26.]. Výsledný narovnaný obraz je použitelný pro pozdější digitalizaci textu. Tímto narovnáním se ušetří spoustu času při pozdější detekci textu na kružnici. Pokud bychom totiž chtěli detekovat text přímo na kružnici, kde je vlastně opsán kolem dokola, bylo by to časově náročnější než detekce textu na přímce. Na kružnici by se musel hledat každý znak, který by byl vlastně různě natočený. To by způsobilo delší práci detekčního algoritmu.

Díky narovnanému textu, který byl původně prohnutý nebo opsaný na kružnici, se k pozdější detekci textu přistupuje jako k normální detekci textu na přímce. Pokud totiž na získaném narovnaném obrazu detekujeme hrany pomocí jedné z výše uvedených metod a takto detekovaný obraz ještě prahujeme například pomocí metody Otsu prahování, je jen malá pravděpodobnost, že bude z takového obrazu problém kvalitně rozpoznat text. Tímto způsobem je v mé aplikaci vyřešená detekce textu opsaného na kružnici.

Negativní vlastností mého algoritmu je možnost současně narovnat pouze jednu kružnici. Takže pokud obsahuje obraz text, který je prohnutý na více místech a kolem různých kružnic, je potřeba takový obraz narovnávat ve více krocích.

4.4.1 Narovnání textu na kružnici

V aplikaci je narovnání považováno jako samostatný modul. Takže pokud bude chtít uživatel narovnávat pouze text na kružnici, přejde na příslušnou kartu nastavení vstupního obrazu, tedy Text na kruhu. Zde je nutno nastavit střed kruhu a poloměr. Pro narovnání je nakonec potřeba nastavit šířku narovnané roviny. Tento postup je vidět na obrázku [Obrázek 29.], kde je znázorněn zeleným křížkem střed kruhu, červenou barvou je ohraničena hranice kruhu. Jakmile jsou tyto dvě hodnoty nastaveny podle požadavků, nastaví se šířka roviny a může se začít narovnávat obraz. Výsledný obraz může být zrcadlově otočený. Z toho důvodu je možné takový obraz otočit zrcadlově. V pravé spodní části okna je vidět výsledek metody narovnání textu na kružnici po aplikaci zrcadlového otočení.



Obrázek 29.: Příklad práce s textem na kružnici

4.5 Možnosti volby relevantních oblastí

Základem pro možností volby relevantních oblastí v aplikaci jsou dvě třídy. Tou první je třída, která představuje bod, CPoint, a tou druhou je třída, která představuje obdélník, ohraničující Vámi vybranou oblast. Tato třída se nazývá CRectangle. Obě dvě třídy implementují rozhraní IObject. Základem pro třídu CRectangle, představující obdélník je první třída, která představuje body, a tedy vrcholy, obdélníka. V následujících kapitolách popíšu potřebné třídy pro výběry relevantních oblastí, kterými jsou výše zmíněné třídy CPoint a CRectangle a navíc ještě třída ObjectDictionary, která představuje objekty pro jednotlivé vstupní obrazy.

4.5.1 Třída CPoint

Jedná se o třídu, která v aplikaci představuje levý horní a pravý dolní bod každého obdélníka. Pokud je tedy obdélník nakreslen na obraze, jsou zde také vytvořeny dva body z třídy CPoint, které jsou vytvořeny na pozicích, zmíněných v předchozí větě. Třída CPoint je doplněna o vlastnost, oznamující, zda byl obdélník vybrán nebo ne. Pokud byl totiž obdélník vybrán, pak jsou vybrány také jeho dva body. To je oznámeno uživateli ohraničením těchto dvou bodů. Stejné oznámení je použito také pro vstup kurzoru myši do oblasti, ohraničené obdélníkem.

Třída má jeden konstruktor, s argumenty představující x-ovou a y-ovou souřadnici bodu a s argumentem, představujícím barvu bodu. Tím je dosaženo vytvoření potřebných bodů vždy při nakreslení nového obdélníka, ale také možnosti umístit na kreslící plochu pouze samostatný bod. Tato volba ale není v aplikaci z rozumných důvodů povolena.

Pro nakreslení bodu je ve třídě CPoint metoda Draw, s argumentem typu Graphics. Tato metoda obvykle vykreslí bod na určitou pozici. Pokud je ale u příslušného bodu změněna vlastnost, která oznamuje, že byl právě tento bod označen, pak je navíc okolo bodu zakreslen kruh, představující určitou hranici bodu, ve které je příslušný bod použitelný. Takže po zachycení bodu v této hranici můžete měnit jeho pozici. Tímto způsobem je jednoduše oznámeno uživateli, které body jsou vybrány a tedy aktivní.

Další metodou ve třídě CPoint je metoda, pro zjištění, zda se kurzor myši nachází v hraniční vzdálenosti bodu. Metoda se nazývá CheckMouseHover a jejím návratovým typem je proměnná typu bool. Argumenty metody představují x-ovou a y-ovou souřadnici kurzoru myši. Na začátku celého algoritmu probíhá kontrola, zda se změnila pozice bodu od předchozí kontroly. Pokud se pozice nezměnila, pak je pouze oznámeno, že se kurzor myši nachází v hraniční vzdálenosti bodu. Ovšem pokud se pozice změnila, je tato nová pozice uložena také v aktuálních souřadnicích vybraného bodu. Zbytek je stejný jako v případě nezměněné pozice.

Třída obsahuje také metodu pro zjištění kliknutí tlačítkem myši na bod, s názvem MouseClick. V této metodě je pouze nastaveno, jestli bylo kliknuto na bod nebo ne. Takže se nastaví proměnná, představující vybrání bodu na příslušnou hodnotu.

Je potřeba také ošetřit, zda je stále stisknuto tlačítko myši nebo ne. K tomu složí metoda MouseButtonStateChanged, která má jeden argument, představující stisknutí tlačítka myši. V této metodě se také nastavuje podle stisknutí tlačítka myši poslední pozice myši. Pokud totiž nebylo stisknuto tlačítko myši, je jeho poslední pozice nastavena na hodnotu null.

Poslední metodou ve třídě CPoint je metoda pro změnu pozice bodu. Tato metoda se jmenuje ChangePosition a obsahuje 2 argumenty, které představují x-ovou a y-ovou souřadnici bodu. V této metodě se tedy nastavuje pozice bodu podle předaných argumentů.

Tímto byla popsána třída CPoint, která představuje levý horní a pravý dolní bod obdélníka.

4.5.2 Třída CRectangle

Třída CRectangle v aplikaci představuje obdélník, který ohraničuje Vámi vybranou oblast obrazu. Tento obdélník má dva aktivní body, které jsou popsány v předchozí kapitole. Třída CRectangle

obsahuje navíc oproti třídě CPoint název objektu, barvu objektu a potřebné údaje pro ukládání hodnot, získaných při fázi OCR.

Třída má dva konstruktory. První konstruktor obsahuje celkem 5 argumentů. Prvním argumentem je název obdélníku, druhý a třetí argument představují levý horní a pravý dolní bod obdélníku. Čtvrtý argument představuje barvu obdélníku a poslední argument představuje výplň obdélníku. V konstruktu jsou nastaveny tyto předané hodnoty hodnotám aktuálního obdélníku a navíc jsou vytvořeny dvě instance třídy CPoint, kterým jsou předané potřebné argumenty dvou bodů a příslušná barva bodu.

Druhý konstruktor třídy CRectangle je speciálně určen pro fázi OCR. Oproti prvnímu konstruktoru obsahuje navíc definici textu, velikost textu a typ písma, získaných při fázi OCR. Pokud obdélník v této fázi neohraničuje textovou část, pak může obsahovat obrázek, který může být také ohraničen tímto typem obdélníku.

Pro nakreslení obdélníku je ve třídě CRectangle metoda Draw s argumentem typu Graphics. Tato metoda slouží pro vykreslení obdélníku na nastavenou pozici. Opět ale obsahuje vyhodnocení situace, kdy je obdélník označen, pokud byla nastavena příslušná vlastnost. Takové označení vypadá o obdélníku jinak než u třídy CPoint. U obdélníku je při označení označen celý obdélník a nad ním je navíc ještě vypsán jeho název. Stejně jako můžete pohybovat s bodem třídy CPoint, tak taky můžete měnit velikost obdélníku změnou pozice jeho dvou aktivních bodů. Samozřejmě můžete měnit pozici také celého obdélníku.

Třída CRectangle obsahuje také metodu CheckMouseHover, podobnou z třídy CPoint. Opět je jejím návratovým typem proměnná typu bool. Tato metoda slouží pro určení, zda se kurzor myši nachází v hraniční vzdálenosti celého obdélníku. Argumenty metody představují x-ovou a y-ovou souřadnici kurzoru myši. Na začátku celého algoritmu je provedena kontrola, zda se změnila pozice bodů odbélníka od předchozí kontroly. Pokud se pozice nezměnila, pak je pouze oznámeno, že se kurzor myši nachází v hraniční vzdálenosti bodu. Ovšem pokud se pozice změnila, je tato nová pozice uložena také v aktuálních souřadnicích vybraného bodu. Zbytek je stejný jako v případě nezměněné pozice.

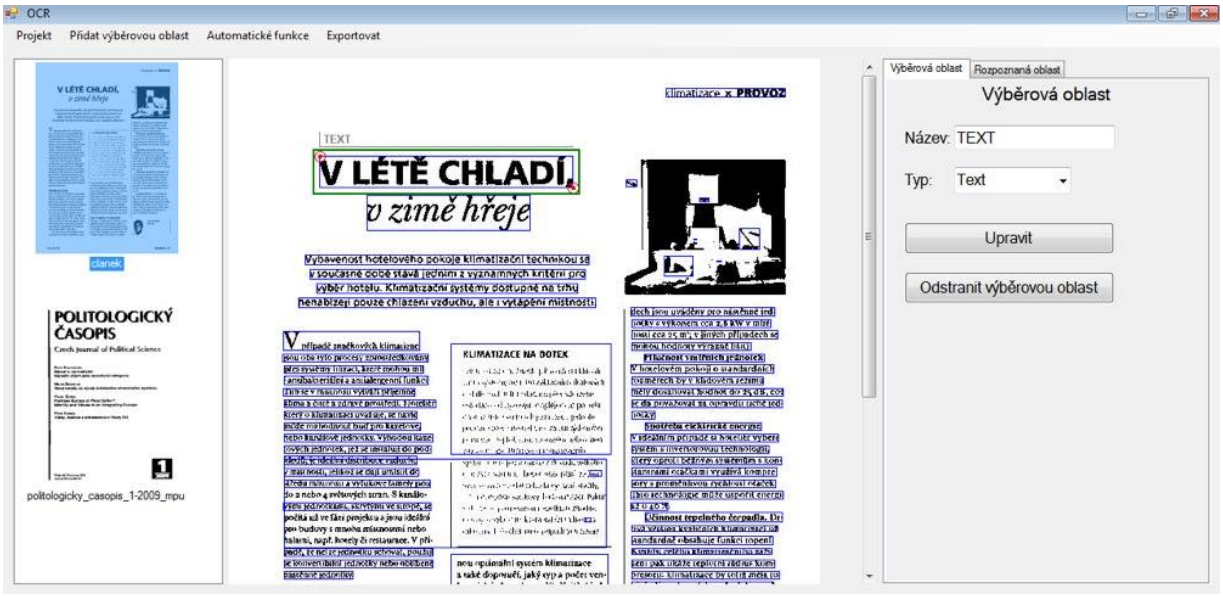
Metody MouseClick a MouseButtonStateChanged jsou identické s metodami z třídy CPoint. Třída CRectangle oproti třídě CPoint obsahuje metodu ChangeObject s dvěma argumenty typu Point, pro změnu pozice celého obdélníku. Tato metoda používá metodu ChangePosition z třídy CPoint na v konstruktoru vytvořené instance objektu CPoint.

4.5.3 Třída ObjectDictionary

U problematiky současných úprav více souborů jsem byl nucen přidat třídu ObjectDictionary, která v programu představuje kolekci pro každý zpracovávaný obraz. Pokud totiž umístíte na obraz výběrový obdélník, je také vytvořena kolekce typu List, do které se ukládají tyto obdélníky. Ty se ale ukládají pouze pro aktuálně vybraný obraz. Z tohoto důvodu se zde nachází třída ObjectDictionary. Ta slouží pro sdružování všech kolekcí typu List pro každý obraz v projektu.

Pokud je tedy vytvořena první kolekce typu List, je také vytvořen objekt třídy ObjectDictionary. Tato třída obsahuje kolekci typu Dictionary [9], která sdružuje všechny kolekce typu List podle názvu všech souborů v projektu. Takže pokud je vyžádána kolekce pro nově vybraný obraz, je vrácena buď

již existující kolekce List, nebo úplně nová kolekce List. Zároveň je uložena kolekce List do kolekce Dictionary podle aktuálního obrazu.



Obrázek 30.: Příklad výběru relevantní oblasti

4.5.4 Možnosti výstupu programu

V předchozích třech kapitolách bylo popsáno, jak se mohou v programu vybírat potřebné oblasti pro každý obraz. Pokud tedy budete mít vybrané a rozpoznané všechny oblasti, které budete chtít uložit do konečné podoby, existuje v programu možnost exportovat všechny hodnoty z Vašich oblastí do formátu XHTML.

Jak bylo zmíněno výše, třída CRectangle může obsahovat buď veškeré potřebné údaje z textové rozpoznané oblasti, nebo vybraný obrázek. Třída ExportSource tedy pracuje s těmito údaji.

Při zahájení exportu Vašeho projektu do formátu XHTML je vytvořena hlavní exportní WWW stránka pro projekt. Z této WWW stránky se můžete později dostat na všechny exportované obrazy z Vašeho projektu. [10]

Export všech jednotlivých obrazů dále probíhá v pořadí, ve kterém byly vloženy výběrové oblasti příslušného obrazu do kolekce typu List. Pokud představuje výběrová oblast rozpoznaný text, je tento text uložen s příslušnou velikostí a s příslušným typem písma do WWW stránky aktuálního obrazu. Pokud ovšem obsahuje výběrová oblast obrázek, je tento obrázek uložen podle specifického jména do složky s obrázky exportovaného projektu. Tento obrázek je nakonec uložen do WWW stránky aktuálního obrazu s příslušnými velikostmi, propočítanými podle velikosti výběrové oblasti.

Tento proces pokračuje tak dlouho, dokud není export proveden ze všech výběrových oblastí z obrazů v projektu.

4.6 Získání zdrojů pro projekt

Kromě použití obrazových souborů typu jpg jako zdroje pro Váš projekt můžete použít dvě další metody pro získání obrazového zdroje pro projekt. Můžete získat zdroj oskenováním přímo v programu, nebo převedením všech stran pdf souboru do jednotlivých jpg souborů.

Oskenování dokumentu je v programu prováděno pomocí třídy WindowsImageAcquisition, která u Windows aplikací komunikuje se skenerem a tudíž dokáže získat oskenovaný obraz. Samozřejmostí je funkční a připojený skener.

Pro převod pdf dokumentu do jpg souborů je použito třídy GhostScript, díky které se uloží každá stránka pdf dokumentu do jpg souboru.

Těmito kroky je sjednoceno použití vstupních formátů na jediný vstupní formát, kterým je obrazový formát jpg.

4.7 Hromadné zpracování dokumentů v programu

Pokud se rozhodnete provádět v programu hromadné zpracování, takže stanovíte pouze vstupní soubory a následně získáte potřebné údaje pro export vstupních souborů, musíte tuto možnost nastavit na obrazovce nastavení v programu.

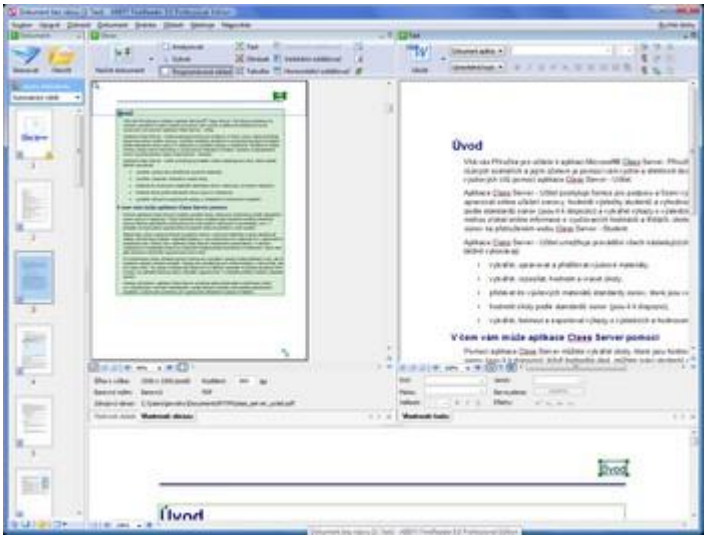
Ve výchozím nastavení je program v režimu veškerých ručních úprav. Takže pokud nechcete provádět ručně veškeré úpravy všech obrazů do požadovaných formátů pro fázi OCR, můžete nastavit provádění úprav a nalezení textů přímo na programu. Tím odpadne nutnost provádět všechny úpravy ručně a ušetří se tím spousta času. Ovšem takové ušetření času si bere určitou daň, v podobě možného nekvalitního převodu obrazu. Ať už jde o nekvalitně zvolenou metodu prahování nebo nevhodně natočeného obrazu.

Pokud tedy neobsahuje Váš projekt velké množství vstupních souborů nebo obsahuje soubory se zakřiveným textem, doporučuji provádět veškeré úpravy ručně. Předejdete tak nevhodným výsledkům, které by Vám znepříjemňovaly práci s programem.

5. Abbyy FineReader

Při samotné implementaci mého nástroje jsem vycházel z funkčnosti komerčního nástroje Abbyy FineReader. Tento program je aktuálně dostupný ve verzi 10, ale v době, kdy jsem prováděl implementaci, byla na trhu dostupná verze 9.

Abbyy FineReader 9 je dostupný ve dvou verzích, Professional Edition a Corporate Edition. Základní vlastnosti jsou stejné, Corporate Edition však přidává integraci do síťového firemního prostředí podporou síťových skenerů a MFP zařízení, síťovou instalací, sdílenými slovníky nebo distribuovaným zpracováním dokumentů. Mezi hlavní výhody programu patří především podpora 179 jazyků, z nichž pro 36 je k dispozici rozsáhlý slovník, výrazně zlepšující rozlišovací schopnosti softwaru. Další výhodou je přístup k dokumentu jako celku (technologie ADRT), z čehož plyne zachování jeho struktury, případně i jednoduššího formátování.



Obrázek 31.: Obrazovka programu Abby FineReader 9

Postup práce s programem Abbyy FineReader spočívá ve třech krocích. S pomocí skeneru, Screenshot Readeru nebo ze souboru (např. PDF, obrázek atd.) získáte data pro rozpoznání. Necháte je zpracovat a otevřete buď ve vlastní aplikaci, nebo v nějakém jiném editoru. Pracovní plocha se pak rozdělí na dvě části, původní náhled rozpoznávaného dokumentu (skenu, PDF) a výsledný text. Program si velmi dobře poradí jak s formátováním, tak i s čárovými kódy, stylem písma atd. Automatická je detekce jazyka, lze ho ale vybrat i individuálně. U upraveného textu dokáže program zkontrolovat pravopis a výsledek převést pro další úpravy do Wordu, Excelu, PowerPointu (podpora formátu Office 2007), Acrobatu, WordPerfectu nebo webového prohlížeče. Program však dokáže přímo generovat i zabezpečené PDF dokumenty.

5.1 Abbyy FineReader Engine

Díky nástroji Abbyy FineReader Engine, můžete za určitý poplatek implementovat vlastní OCR programy s funkčností programu Abbyy FineReader. Tento nástroj si můžete vyzkoušet v 60-ti denní verzi zdarma, nebo si ho můžete za nemalou částku pořídit. Vysoká cena nástroje je z důvodu poskytnutí veškeré funkčnosti z programu Abbyy FineReader 9. Díky této funkčnosti můžete později produkovat vlastní OCR programy s nutností minimálních znalostí problematiky OCR. Získání 60-ti denní verze je podmíněno podpisem smlouvy s firmou Nupseso CZ, s r.o., která je dodavatelskou firmou softwarů firmy Abbyy, o neposkytnutí dodané 60-ti denní verze třetí straně.

5.2 Porování

Již na začátku implementace se můj program s programem Abby FineReader rozchází. Nevýhodou programu Abby FineReader je nutnost nastavení typu nového projektu. Takže musíte přesně specifikovat vstupní formát, se kterým bude program po dobu práce v novém projektu pracovat. Vstupními formáty programu jsou obrázky, pdf soubory nebo oskenované dokumenty. V mém programu se taková specifikace pro nový projekt nemusí provádět, díky převodu vstupních souborů do jednotného formátu jpg.

Výhodou programu Abbyy FineReader je několikaletý vývoj celé aplikace a tím pádem zlepšování celého programu pro co nejkvalitnější získání výsledku rozpoznání. Také možnost automatického rozpoznání jazyka je výhodou.

Existencí slovníku pro rozpoznávané jazyky je možnost kvalitního rozpoznání dále zvýšena. Také já jsem do své aplikace zařadil slovník. Ovšem pouze český slovník. Takže při každém rozpoznávání jsou porovnávány všechny nalezené slova se slovy ze slovníku, navíc s tou možností, že můžete špatně rozpoznané slova tzv. „přidat do slovníku“ a tím pádem naučit správného rozpoznání pro pozdější nalezení takových chybných slov.

Program Abbyy FineReader má možnost exportovat Vámi rozpoznané soubory do více formátů. Oproti tomu můj program provádí export pouze do formátu XHTML, takže pro zobrazení exportovaných souborů z mého programu musíte mít navíc nainstalován na Vašem počítači internetový prohlížeč.

Má práce rozšiřuje možnosti OCR také pro rozpoznávání prohnutého textu. Takže oproti programu Abbyy FineReader nabízí mé řešení metodu rozpoznávání textu na kružnici, která je popsána výše. Tímto mé řešení nabízí možnost rozpoznání ne pouze vodorovně, případně horizontálně orientovaného textu, ale také text obtočený kolem kružnice.

6. Závěr

V této práci jsem popsal možnosti úpravy vstupních materiálů a problematiku OCR. Práce zahrnuje problematiku úprav nedokonalostí a vad vstupních materiálů, popsání některých možných řešení rozpoznání textu, až po jednu z možností rozpoznání prohnutého textu.

Součástí mé práce je také aplikace, která řeší veškeré popsané problémy úprav vstupních materiálů, zabývá se také problematikou rozpoznání prohnutého textu a v poslední řadě řeší problém rozpoznání tištěného textu, takže OCR.

Realizace tohoto projektu si vyžádala seznámení si s problematikou OCR a s problematikou úprav vstupních materiálů pro OCR.

S praktickou realizací podobného projektu jsem dosud neměl žádné zkušenosti, proto jsem nejprve prostudoval potřebné články a knihy, které se věnují problematice OCR.

Důkladnost prostudování jednotlivých metod pro úpravu vstupního obrazu se velmi výrazným způsobem projevila při implementaci, kdy se čas věnovaný prostudování mnohonásobně vrátil.

Tento projekt bych chtěl dále rozvinout na možnost automatického rozpoznání jazyka a dále snížit interakci uživatele s programem, pro získání rozpoznaného textu na minimum. Díky možnosti automatického rozpoznání jazyka by bylo možné používat slovníky pro korekturu rozpoznaných slov také pro jiné jazyky než pro češtinu. Pokud bych chtěl snížit interakci uživatele s programem, pro získání rozpoznaného textu na minimum, musel bych zařadit potřebné detekční algoritmy, které automaticky rozpoznají nalezený text na kružnici a příslušný poloměr a střed této kružnice. Tím bych eliminoval nutnost zásahu uživatele do programu při rozpoznávání textu na kružnici a tedy jedinou část, u které musí programu přímo asistovat uživatel.

Literatura a informační zdroje

Internetové zdroje

[1] Technet.cz [online]. 2007 [cit. 2010-04-26]. Jak se počítač naučil číst milion knížek ročně. Dostupné z WWW: <http://technet.idnes.cz/jak-se-pocitac-naucil-cist-milion-knizek-rocne-fo8-/tec_technika.asp?c=A071123_182221_tec_technika_pka>

[2] Čárový kód In Wikipedia : the free encyclopedia [online]. St. Petersburg (Florida) : Wikipedia Foundation, 4.7.2006, 21.4.2010 [cit. 2010-04-26]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Čárový_kód>

[3] Chip.cz [online]. 2009 [cit. 2010-04-26]. Papírové dokumenty v počítači. Dostupné z WWW: <http://www.chip.cz/clanky/trendy/2009/9/papirove-dokumenty-v-pocitaci>

Knižní zdroje

[4] DOBEŠ, Michal. Zpracování obrazu a algoritmy v C#. Praha : BEN - technická literatura, 2008. 144 s. ISBN 978-80-7300-233-6.

[5] CHEN, Qing. Evaluation of OCR Algorithms for Images with Different Spatial Resolutions and Noises. Ottawa, Kanada, 2003. 122 s. Diplomová práce. University of Ottawa

[6] SOJKA, Eduard. DIGITÁLNÍ ZPRACOVÁNÍ A ANALÝZA OBRAZU. Ostrava, 2000. 133 s. Skripta. VYSOKÁ ŠKOLA BÁNSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA. Dostupné z WWW: <http://mrl.cs.vsb.cz/people/sojka/digitalni_zpracovani_obrazu.pdf>.ISBN 80-7078-746-5

[7] PAL, U.; ROY, P.P. Multi-oriented and curved text lines extraction from Indian documents. IEEE Trans.on SMC - Part B. 2004, 34, s. 1676-1684.

[8] VASUDEV, T.; HEMANTHKUMAR, G.; NAGABHUSHAN, P. Transformation of arc-form-text to linear-form-text suitable for OCR. ScienceDirect. 2007, 07, s. 2343–2351.

[9] PETZOLD, Charles. Programování Microsoft Windows Forms v jazyce C#. Prahq : Computer Press, 2006. 360 s. ISBN 80-251-1058-3

[10] PROSISE, Jeff. Programování v Microsoft .NET. Praha : Computer Press, 2003. 736 s. ISBN 80-7226-879-1